Calhoun
Institutional Archive of the Naval Postgraduate School

Calhoun: The NPS Institutional Archive

DSpace Repository

Theses and Dissertations 1. Thesis and Dissertation Collection, all items

1978-06

A microcomputer based shipboard surface-subsurface contact plotter system Pt.1

Goncalves, Antonio Luiz Soares; De la Cuba Bravo, Javier Enrique

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/18496

Copyright is reserved by the copyright owner

Downloaded from NPS Archive: Calhoun

DUDLEY KNOX LIBRARY

Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

http://www.nps.edu/library

A MICROCOMPUTER BASED SHIPBOARD SURFACE-SUBSURFACE CONTACT PLOTTER SYSTEM.
PT.1

Antonio Luiz Soares Goncalves

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

A MICROCOMPUTER BASED SHIPBOARD
SURFACE-SUBSURFACE CONTACT PLOTTER SYSTEM

by

Antonio Luiz Soares Goncalves

and

Javier Enrique De la Cuba Bravo

June 1978

Thesis Advisor: Stephen T. Holl

Approved for public release; distribution unlimited.

U184180

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS
BEFORE COMPLETING FORM

1. REPORT NUMBER

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
A Microcomputer Based Shipboard Surface-Subsurface Contact Plotter System

5. TYPE OF REPORT & PERIOD COVERED
Master's Thesis;
June 1978

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
Antionio Luiz Soares Goncalves
Javier Enrique De la Cuba Bravo

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Naval Postgraduate School
Monterey, California 93940

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS
Naval Postgraduate School
Monterey, California 93940

12. REPORT DATE
June 1978

13. NUMBER OF PAGES
508

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)
Naval Postgraduate School
Monterey, California 93940

15. SECURITY CLASS. (of this report)
Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Microcomputer
Microprocessor
Maneuvering Board
Floating Point Hardware
Plasma Display
Shipboard Computer
DRT
ASW

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Today, in most shipboard Combat Information Centers, true motion plots of own ship's motion and or surface/subsurface contacts are provided with the aid of electromechanical devices like the dead-reckoning tracer ("DRT") or the NC-2 plotter. These devices suffer a variety of drawbacks, such as inflexibility and the need to "track" the light-spots manually. This thesis has developed a flexible, labor-saving, true-motion plotter using contemporary microcomputer and plasma display technologies. An

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

automatic display of contact data, analogous to the plexiglass "surface status board" is also provided. Contact course, speed and CPA information are computed automatically and, unlike the DRT or the NC-2 plotter, both the geographic and tabular displays can be easily duplicated on the bridge and elsewhere.

Approved for public release; distribution unlimited.

A MICROCOMPUTER BASED SHIPBOARD SURFACE-SUBSURFACE CONTACT PLOTTER SYSTEM

[pt. 1]

by

Antonio Luiz Soares Goncalves
Lieutenant Commander, Brazilian Navy
B.S., Brazilian Naval Academy, 1964

and

Javier Enrique De la Cuba Bravo
Lieutenant (JG), Peruvian Navy
B.S., Peruvian Naval Academy, 1974
M.S. Comp. Sys. Mgt., Naval Postgraduate School, 1977

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
JUNE, 1978

ABSTRACT

Today, in most shipboard Combat Information Centers, true motion plots of own ship's motion and or surface/subsurface contacts are provided with the aid of electromechanical devices like the dead-reckoning tracer ("DRT") or the NC-2 plotter. These devices suffer a variety of drawbacks, such as inflexibility and the need to "track" the light-spots manually. This thesis has developed a flexible, labor-saving, true-motion plotter using contemporary microcomputer and plasma display technologies. An automatic display of contact data, analogous to the plexiglass "surface status board" is also provided. Contact course, speed and CPA information are computed automatically and, unlike the DRT or the NC-2 plotter, both the geographic and tabular displays can be easily duplicated on the bridge and elsewhere.

TABLE OF CONTENTS

LIST OF TABLES 9

LIST OF FIGURES 10

I. INTRODUCTION 11

II. INTRODUCTION TO THE PROBLEM 12

A. USUAL COMBAT INFORMATION CENTER OPERATIONS .. 12

B. MANEUVERING BOARD PLOTTING SHEETS 12

C. DEAD-RECKONING EQUIPMENT 13

III. RECENT MICROPROCESSOR TECHNOLOGY ADVANCES 17

IV. CONCEPTUAL DESIGN 20

A. HARDWARE DESIGN CONSIDERATIONS 21

B. SOFTWARE DESIGN CONSIDERATIONS 22

1. Language Selection 22

2. Floating Point Arithmetic 22

3. Transcendental Functions 24

4. Design Philosophy 24

5. User Interface 27

6. Levels of Correctness 28

V. SYSTEM DESCRIPTION 31

A. HARDWARE DEPENDENCIES 31

B. SYSTEM CHARACTERISTICS 33

1. System Displays 35

a. Video Terminal Display 35

(1) Input Mode 40

(2) Display Mode 41

(3) Other Function-Defined Keys 43

b. Plasma Display........................ 43

C. DATA STRUCTURES.............................. 46

1. Data Structure for the System Parameters.............................. 48

2. Data Structures for Own Ship Parameters.............................. 50

3. Data Structures for the Contact Parameters.............................. 53

VI. CONCLUSION.................................... 58

A. RESULTS.................................... 58

B. FUTURE WORK.................................... 58

APPENDIX A ALGORITHM DESCRIPTION...................... 60

A. THE CLOSEST POINT OF APPROACH (CPA)......... 60

1. The Basic Relative Movement Problem..... 60

2. Other Uses of the Maneuvering Board..... 60

3. Maneuvering Board Problems Solutions Implemented.................... 62

4. Problems that Normally Occur............ 62

5. The Least-squares Fit Approach.......... 62

6. CPA Algorithm........................... 65

B. GRAPHICS ON PLASMA DISPLAY.................. 70

1. Physical Considerations................. 70

2. Plasma Display Unit Capabilities........ 70

3. Algorithm Design........................ 71

a. Windowing........................ 71

b. Procedure to Check if a Given Position Falls Within the

Limits of the Defined "Window"...... 71

c. Normalization........................ 73

d. Plasma Reorientation................ 74

C. TRANSCENDENTAL FUNCTIONS..................... 75

1. Cosine and Sine Functions............... 76

2. Arc Tangent Function..................... 77

D. POSITIONAL DATA CONVERSION................. 78

1. Convert LAT and LONG to X/Y Coordinates............................ 79

2. Convert a Given Position in Terms of Bearing and Range from Own Ship to X/Y Coordinates................. 79

3. Convert X/Y Coordinates of a Given Position Into Latitude and Longitude..... 80

APPENDIX B SOFTWARE CATEGORIZATION................... 81

A. MODULES DESCRIPTION........................ 81

B. MODULES INTERACTION........................ 81

APPENDIX C OPERATOR'S MANUAL........................ 85

APPENDIX D FLOATING-POINT HARDWARE BOARD............. 134

A. GENERAL INFORMATION........................ 134

B. DESCRIPTION OF THE MATH UNIT............... 134

C. PREPARATION FOR USE........................ 136

1. Installation Considerations............. 136

2. I/O Base Address Switches............... 136

3. Programming Information................. 137

4. Math Unit Functions..................... 139

5. Argument and Result Data Formats........ 139

6. Status and Flags........................ 141

7. Examples of Floating-Point Number

Representation........................... 147

APPENDIX E PROGRAM LISTINGS........................... 149

A. EXTERNAL DECLARATIONS........................ 149

B. PROCEDURES BY MODULE........................ 149

C. LISTINGS................................... 156

LIST OF REFERENCES................................... 505

INITIAL DISTRIBUTION LIST............................ 507

LIST OF TABLES

I. MATH UNIT FUNCTIONS AND EXECUTION TIMES........ 25

II. FORMATS AND UNITS USED........................... 36

III. CONVERSION FACTORS USED......................... 37

IV. I/O ADDRESSING................................... 138

V. ARITHMETIC AND CONVERSION FORMATS................ 140

VI. OPERATION ARGUMENT AND RESULT DATA FORMATS..... 142

VII. FLAG BYTE FORMAT................................ 143

VIII. STATUS BYTE FORMAT............................. 143

IX. FLOATING-POINT NUMBERS........................... 148

LIST OF FIGURES

1. DRT BASIC COMPONENTS.......................... 14

2. MK NC-2 MOD 2 PLOTTING SYSTEM.................... 16

3. EQUIPMENT CONFIGURATION.......................... 23

4. VIDEO TERMINAL DISPLAY PICTURE ARRANGEMENT..... 38

5. KEYBOARD LAYOUT................................ 44

5a. INPUT AND DISPLAY KEYS ARRANGEMENT............ 45

6. PLASMA DISPLAY VIEW............................ 47

7. DATA STRUCTURE FOR THE SYSTEM PARAMETERS....... 49

8. DATA STRUCTURES FOR OWN SHIP PARAMETERS........ 51

9. DATA STRUCTURES FOR CONTACT PARAMETERS......... 54

10. MANEUVERING BOARD SCHEMATIC.................... 61

11. USUAL PLOTTING AT THE MANEUVERING BOARD........ 63

12. SPECIAL CASES.................................. 66

13. WINDOWING SCHEMATIC............................ 72

14. EQUIPMENT VIEW................................. 86

I. INTRODUCTION

A rapidly developing and expanding area of computer technology and application is that of microprocessors; paralleling the advances in microprocessor technology, the manufacturers of graphic displays are making steady progress in expanding their local capability, while at the same time reducing their cost.

Because a significant portion of the cost of building or maintaining a warship is the electronics in its sensor and weapons systems, and because a great amount of time and personnel power is employed in performing simple but important tasks, microcomputers offer the potential to: (1) reduce the cost of digital systems, (2) perform some complex functions at remote stations, relieving the congestion at larger central computing facilities, and (3) perform functions currently handled by watch personnel, thus reducing the manning requirements of watch sections. An example of this is the problem of manual tracking of radar contacts and the solution of Maneuvering Board problems.

It is the purpose of this thesis to demonstrate that an alternate approach to the solution of the problems mentioned above can be developed and implemented by using a microcomputer plasma graphics system, while at the same time maintaining a human engineered user interface.

II. INTRODUCTION TO THE PROBLEM

On board naval ships not equipped with Naval Tactical Data Systems (NTDS), operations performed by the Combat Information Center (CIC) during a normal peacetime watch include manual tracking of radar contacts and solution of maneuvering board problems.

A. USUAL COMBAT INFORMATION CENTER OPERATIONS

During normal peacetime steaming, the CIC watch team may consist of from two to ten or even more personnel, depending on the size of the ship as well as on the complexity of the equipment being used.

Among the problems that are normally solved by CIC personnel, special mention needs to be made of those of plotting contacts and the determination of parameters such as course, speed and closest point of approach (CPA) of those contacts. This is a tedious and error-prone task; it often requires most of the time and effort of the CIC team, and it is vitally important to the safety of the ship. This has lead to the installation of equipment to reduce the amount of workload in the CIC while at the same time improving the reliability of the constant information provided to the bridge; this equipment includes dead-reckoning devices and the NC2 plotter.

B. MANEUVERING BOARD PLOTTING SHEETS

The primary responsibility of CIC is to provide

information and recommendations on the tactical situation. Accordingly, CIC must supply information on all surface contacts within range. Contact course, speed and CPA information, is usually found using the "Maneuvering Board" plotting sheets.

The Maneuvering Board plotting sheet (H. O. 2665-10) has been prepared in order to facilitate the solution of a ship's relative movement problem.

Although the use of the Maneuvering Board becomes after some practice, straightforward, it will normally require the complete attention of one person during CIC operations.

C. DEAD-RECKONING EQUIPMENT

Dead-reckoning equipment is an important device in CIC. It can maintain a continuous, up-to-the-minute, geographic plot of own ship's motion.

The dead-reckoning system consists of the following basic components: (1) dead-reckoning analyzer (DRA), (2) dead-reckoning indicator (DRI), and (3) dead-reckoning trace (DRT), which are shown in Figure No. 1. Course and speed inputs from the own ship are fed into the DRA from the gyrocompass and pitometer log, and then to the DRT, where they cause a movable source of light to trace the ship's track continuosly.

The chief value of the DRT is its use in analyzing ship movements and in planning and carrying out maneuvers. As a geographic plotting device, the DRT displays true courses and allows direct computation of true contact speeds.

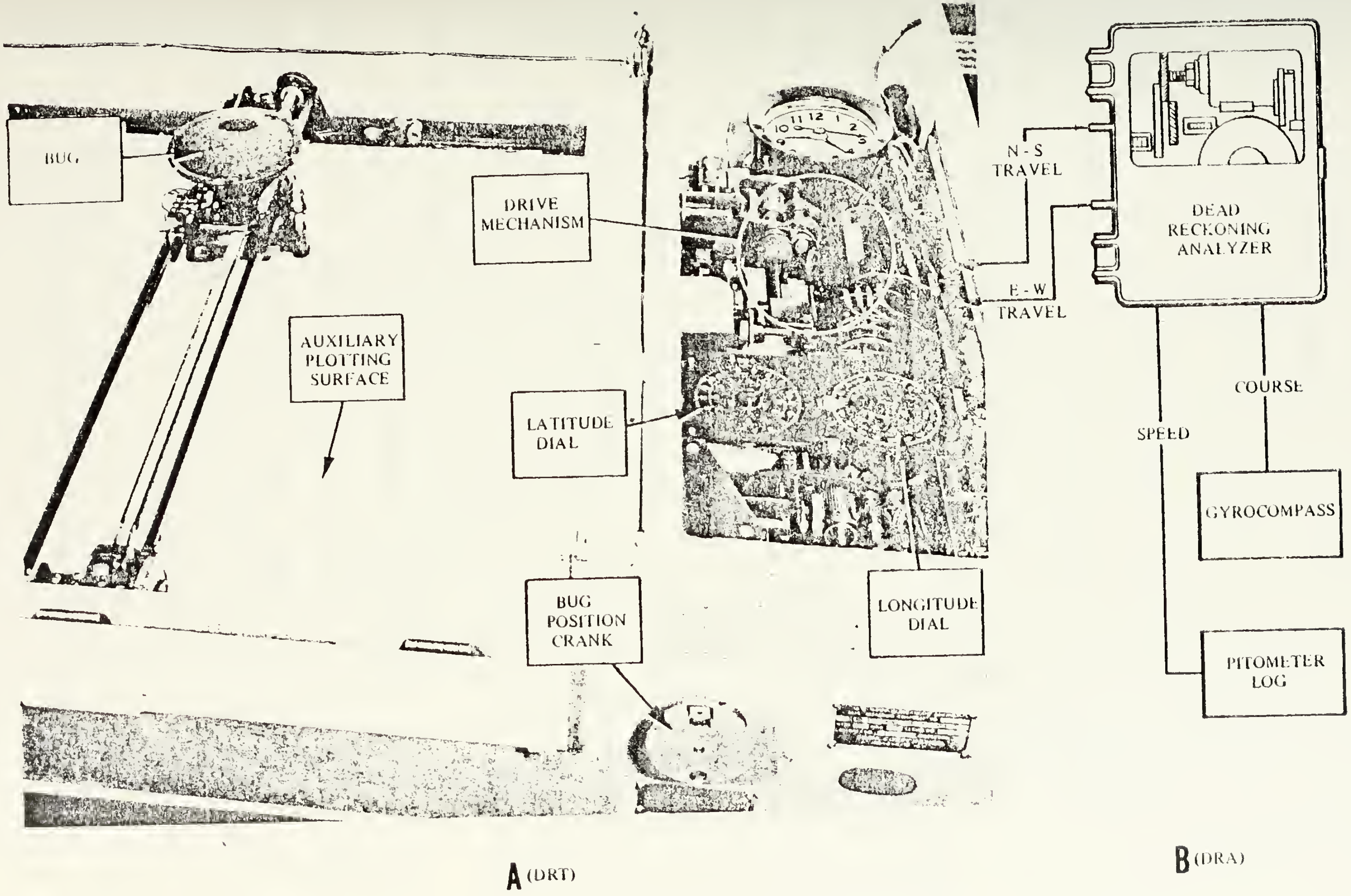


FIGURE NO. 1 DRT BASIC COMPONENTS

Marking positions of the bug indicates true positions of own ship; connecting these plotted positions yields the ship's track. Plotting ranges and bearings of contacts, using own ship's positions as references, establishes their true positions. An experienced DRT operator can maintain simultaneous plots of as many as half a dozen contacts, meanwhile supplying essential data (as required) on contacts that are being plotted.

A problem that is present, especially in Anti Submarine Warfare (ASW), is that of plotting fast moving ships, and tight-turning submarines, which normally results in confused and inaccurate DRT plots. In order to help solve these problems, as well as to reduce inconvenient delays in plotting contacts because of information relayed through phone talkers, the USN Mk NC2 plotting system (Canadian DRT) was developed. The NC2 plotter consists of three major units: (1) the plotting table, (2) the dead-reckoning indicator, and (3) data converter.

The main difference between the NC2 plotting system and the DRT is that the NC2 plotting system is capable of receiving contact bearing and range information directly from four sources (usually 3 radar repeaters and the sonar). This information is then translated, and presented as colored points of light in the plotting table. A functional diagram of the NC2 plotting system is presented in Figure No. 2.

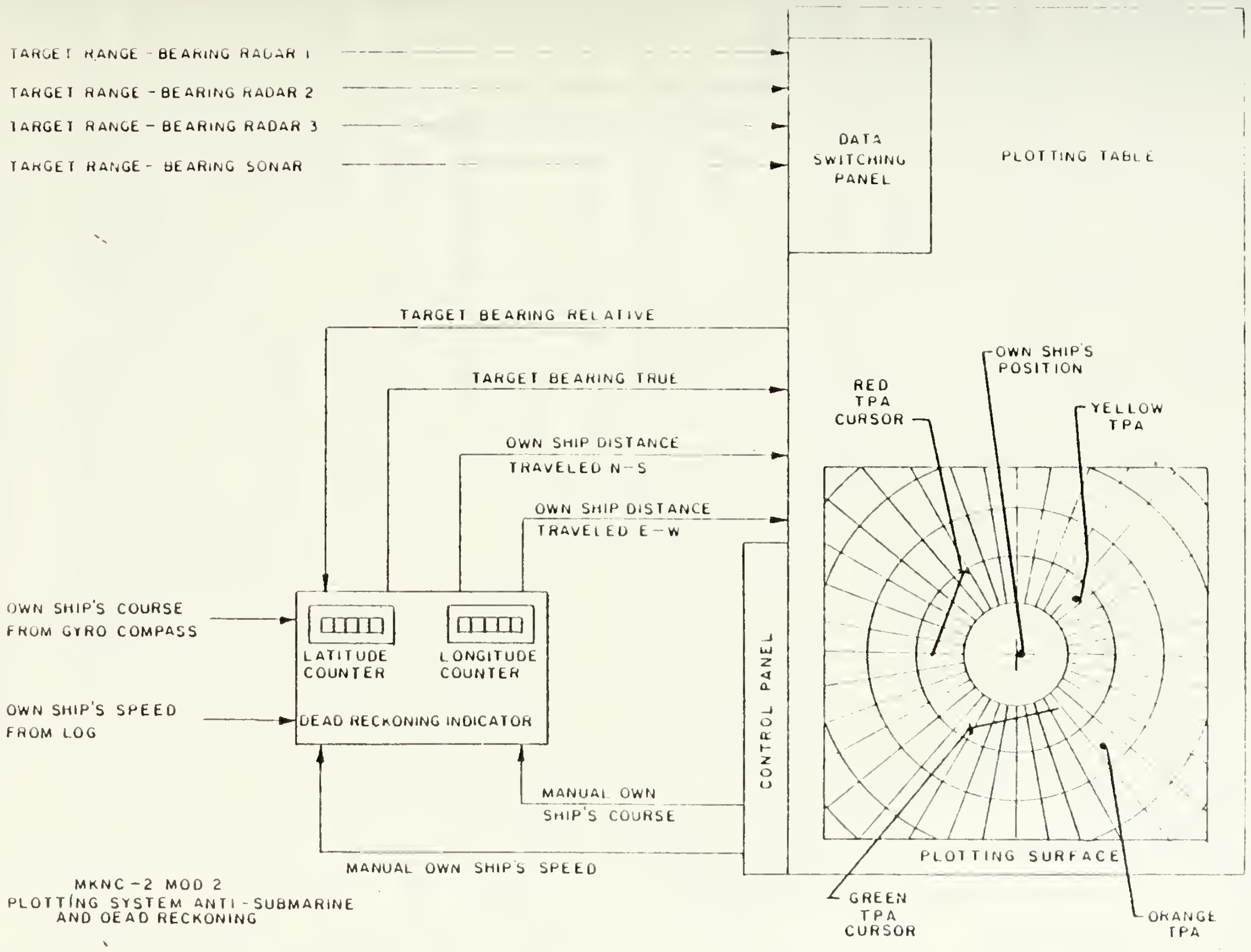


FIGURE NO. 2 MK NC-2 MOD 2 PLOTTING SYSTEM

III. RECENT MICROPROCESSOR TECHNOLOGY ADVANCES

Recent advances in the field of large-scale integration LSI) semiconductor process technology have made possible substantial reductions in the cost and size of digital logic circuits. In the last decade, computer system building blocks have progressed from discrete components to complex integrated circuits. Microprocessors represent a very remarkable achievement of engineering ingenuity and industrial know-how at their best.

Composed of miniaturized software-encoded chips and minimal hardware circuitry, the microcomputer, a digital computer using microprocessor logic, is ideally adapted to specialized tasks and applications.

Since 1973, the year in which INTEL Corporation shipped the first 8080 8-bit N-channel microprocessor, a number of manufacturers have developed similar products and because of this competition prices have been drastically lowered. With hardware and software system development costs as they are, much micro work is done on a custom basis. The greatest advantage of microcomputer based systems is the ability to have specialized and dedicated equipment solving specific problems.

At the same time peripheral equipment has also experienced major changes, perhaps most dramatically in graphic displays. The cathode ray tube (CRT) is the most common device because of its high performance, low cost, and

quality.

A relatively new graphic device is the Plasma Display. A plasma panel is two etched glass plates separated by a neon-based gas which glows when excited by an electric pulse. The display consists of a series of bright dots that can be formatted into alphanumeric and graphic symbols. Plasma panels do not require refresh procedures and once a particular point on the display is "turned on", it continues to glow until "turned off".

The simplicity of construction of the plasma panel suggests that it can potentially replace the CRT for many computer graphic applications. In its current state of development, it presents the following advantages: (1) it presents a sharper image that does not deteriorate with time (does not need to be refreshed), (2) it has a reliable selective erasure mechanism, (3) its power requirements are comparatively low, (4) it has a longer expected life time, and (5) it occupies less space. Some disadvantages are: (1) lower resolution, (2) relatively slow write and erase rates, and (3) no "real" gray scale [Ref 11].

As O. Babin and R. Seaman have shown [Ref. 1], it is possible to combine the advantages and convenience of low-cost microcomputer systems, with the powerful technology of plasma displays, thus having a complete microcomputer development system with graphics capability.

The purpose of this thesis was to demonstrate the applicability of a simple and inexpensive microcomputer

plasma graphics system to Navy functions and problems, specifically as an alternate approach to the solution of CIC's most common problems, and to show that it is possible to design and develop a human engineered user interface.

IV. CONCEPTUAL DESIGN

Microprocessor technology can be used to solve Maneuvering Board problems and to present a geographical plot of own ship and contact positions.

In order for such a system to be competitive with existing equipment, the following requirements should be fulfilled:

1. Relatively inexpensive acquisition and maintenance costs.
2. Reliable and widely available components.
3. Speed and accuracy in performing necessary calculations.
4. Human engineered user interface.
5. Flexibility in its use.
6. Capacity of displaying a geographic plot of the own ship and contacts, while maintaining at the same time, a constant update of the own ship's position.
7. Capacity to solve Maneuvering Board problems such as the determination of CPA information, and calculation of course and speed values for the contacts.
8. Operation of the proposed system should not require more people than the current methods.

It was with these objectives in mind that the microcomputer system described below was designed.

A. HARDWARE DESIGN CONSIDERATIONS

The equipment selected to implement the system described in this thesis was chosen because it was available at the Naval Postgraduate School, and was representative of commercially available microcomputers and plasma display technologies. The selection of the INTEL Microcomputer Develeopment System (MDS) computer as the prototype for this project was made because of its immediate availability at the Naval Postgraduate School and because O. Babin and R. Seaman [Ref 1] had already interfaced the MDS with a plasma display (the AN/UYO-10 Plasma Display Set, manufactured by SCIENCE APPLICATIONS INC (SAI)).

Two features of the MDS system that were especially useful were: (1) the 8-level, nested interrupt priority resolution network, and (2) the real-time clock logic, used to maintain a real time clock value by means of generating an interrupt at 0.77 millisecond intervals.

A DATAMEDIA Elite 2500 Video Terminal was chosen to present alphanumeric information, because its features included: (1) editing and roll operation modes, (2) 50 to 9600 baud programmable speed transmission, (3) protected fields, (4) computer derived or high light field (blink), (5) addressable cursor, and (6) provision to drive up to 16 external monitors.

Because of the volume of floating point calculations required, an SBC 310 High Speed Mathematics Unit, developed by Intel Corporation, was incorporated. In performing high-

speed mathematical functions, the Math Unit acts as an intelligent processor, performing a repertoire of up to 14 arithmetic functions at least an order of magnitude faster than comparable software routines.

Figure No. 3 shows the final configuration of the equipment used in developing this thesis.

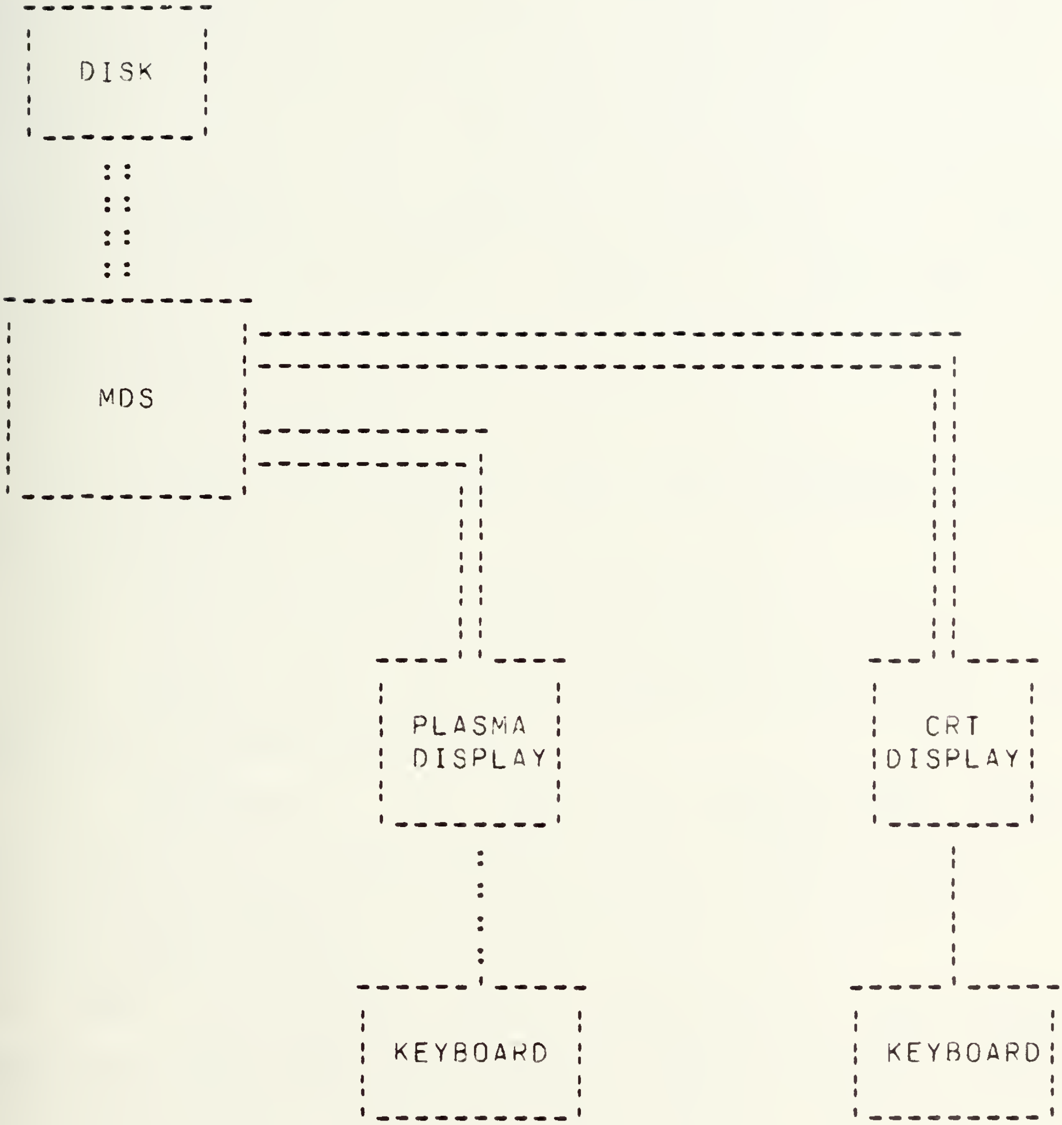
B. SOFTWARE DESIGN CONSIDERATIONS

1. Language Selection

Because of the hardware configuration selected and its immediate availability, PL/M 80 was chosen as the programming language to be used. PL/M 80 is a language developed by Intel Corporation and designed especially for system and applications programming for the Intel 8080 microprocessor. The ISIS-II disk operating system, also developed by Intel Corporation for the Intellec MDS system, has a resident PL/M 80 compiler, which was very useful during the implementation, debugging and testing phases.

2. Floating Point Arithmetic

Floating point arithmetic was required because the range of numbers to be represented was large and sometimes unpredictable. The format used to represent floating point numbers was as required by the SBC 310 High Speed Mathematics Unit. Appendix D shows how this unit was actually implemented and the required software procedures that are needed to use it. The functions that were



NOTE.- .... denotes connection used only for implementation purposes.

FIGURE NO. 3 EQUIPMENT CONFIGURATION

implemented and the execution times are listed in Table I.

3. Transcendental Functions

Three transcendental functions were also implemented, namely: (1) cosine of a given angle in radians, (2) sine of a given angle in radians, and (3) arc tangent of the ratio of two input parameters. These functions were implemented based on the procedures in Reference 4, using floating point procedures. Appendix A describes the algorithms used.

4. Design Philosophy

The system was designed with the following three specific objectives in mind:

a. Human engineered user interface.

b. Capability to display a geographic plot of the own ship and contact positions.

c. Capability of solving Maneuvering Board problems, specifically the determination of CPA information, course and speed values of all contacts.

In order to achieve these objectives, a bottom-up implementation philosophy was chosen. Because of the modular design encouraged by PL/M 80 and ISIS-II, it was also possible to map the different levels of design into corresponding modules of software. The basic idea was to encompass all functions corresponding to a level of design into one software module capable of performing all the necessary functions.

In doing so, the following modules were developed:

OPERATION NAME	OPERATION CODE	TYPICAL EXECUTION TIME	MAXIMIUM EXECUTION TIME
Fixed Point Multiply (MUL)	0	15	20
Fixed Point Divide (DIV)	1	26	30
Extended Fixed Point Div(EDIV)	E	84	100
Float. Point Multiply (FMUL)	2	84	100
Float. Point Divide (FDIV)	3	92	110
Float. Point Add (FADD)	4	33	75
Float. Point Subtract (FSUB)	5	33	75
Float. Point Square (FSQR)	6	84	100
Float. Point Sqr Root(FSQRT)	7	178	205
Fixed-to-Float Cnvrsion(FLTDS)	8	72	100
Float-to-Fixed Cnvrsion(FIXSD)	9	42	85
Float. Point Compare (FCMPR)	A	7	7
Float. Point Test (FZTST)	B	7	7
Exchange (EXCH)	F	4	4

Note: all time values are specified in microseconds. Listed times do not include time to pass arguments to the Math Unit and to read results upon completion: this is typically 90 microseconds.

TABLE I. MATH UNIT FUNCTIONS AND EXECUTION TIMES

a. BASICS: a module used to interface the system with the CRT/keyboard.

b. PLASMA$PRIMITIVES: a module used to interface the system with the Plasma Display unit.

c. TIME: a module used to perform all functions dealing with time, and also to keep a real time clock for the system.

d. FLOATING$POINT: a module used to perform all the necessary floating point operations, and to calculate the transcendental functions already described.

e. FLTASCII: a module used to make the necessary conversions from a string of ASCII characters into floating point format, and vice versa.

f. CRT: a module used to display all the necessary and requested values in the CRT.

g. PLASMA$MODULE: a module used to display all the necessary graphic information in the Plasma Display unit.

h. COMMANDS: a module used to interface with the user to get requested input values.

i. DISPLAY$CMDS: a module used to display information requested by the user, in the CRT.

j. CPA$MODULE: a module used to calculate and solve all CPA values and problems.

k. EXECUTIVE$CMDS: a module containing all the necessary procedures to link all the other modules' actions.

l. MAIN: a module used to serve as executive module of the system by combining all the different functions of

the already described modules.

Appendix B describes all these modules in more detail, while Appendix E lists all the programs that compose the system; Appendix A describes all the main algorithms used in the system, and Appendix C contains an Operator's Manual giving instructions on how to use the system.

5. User Interface

This is the aspect in which most systems fail, because of the complexity of the problem of trying to define what an "average user" is in any system, and in trying to encompass all the possible ways in which a user could react. It is therefore necessary to define boundaries in how the system is expected to interact with the user, while at the same time providing an acceptable range of variations.

One of the features that becomes indispensable when nteracting with a human operator, is error detection and otification as soon as possible, and the capability of llowing the user to correct his own mistakes either when so old by the system, or when he discovers the mistake by imself. Special care was taken in this aspect, as is xplained below.

One decision that was taken during system design was to educe to a minimum the number of keys that the operator ould have to press when entering data. This implied that no pecial character, such as carriage return, would be ecessary to mark the end of an input, but this also meant t the same time that the input would have to be done in a

pre-formatted manner. The design choice of doing all inputs in a pre-formatted way, was reinforced by the fact that personnel in a CIC team are used to communicate with each other using a pre-defined terminology. That is, if the value of the course of a contact is 030 degrees, then when communicating this value, the word "thirty" is not used but instead the message "zero", "three", "zero" is given.

6. Levels of Correctness

As was established before, it is very important for a system that depends almost completely on the correctness of the input values that it receives, to ensure that all the input values received fall into an acceptable range of variation, and within logical and plausible limits; furthermore, the capability of error correction should also be embedded in the system.

To achieve all of this, the system was designed to have up to six levels of error detection and correction:

a. The first level is established by checking for invalid requests from the user, such as asking for information about a contact when no contacts are in the system; if this error occurs, a warning message is issued.

b. The second level of error detection is done by checking each character received from the keyboard, to determine if it has some meaning to the system. An example of this would be when trying to input an alphabetic character in a numeric defined field; if an error of this type is detected, then the CRT's alarm will sound and the

acter will not be echoed at the display.

c. The third level is established by checking a actically correct input against established boundaries the type of input being expected. An example of this ld be when a value of 79 is received when requesting an RS value; since 79 lies outside the boundaries ablished for HOURS (0 <= HOURS < 24) then an error has urred; thus, a warning message will be issued, the CRT's rm will sound, the value will not be accepted nor cessed by the system, and the cursor will be placed at beginning of the incorrect value; the warning message l remain on the screen until the mistake is corrected.

d. The fourth level occurs after all input has been ceived from the keyboard, by giving the user the chance to rrect any value just input.

e. The fifth level is obtained by making the system all the necessary prompting in a pre-determined format. is eliminated the problems that occur when more or less ta than necessary is provided to the system.

f. The sixth level is carried out by allowing the ser to change any current own ship or contact value at any ime.

All these levels of correctness are ensured any time an put operation takes place. It is important to note that no ovision for values that are syntactically correct and ithin established boundaries, but incompatible with evious data and logically incorrect with the present

situation, could be provided because of the many parameters, variables and special situations that could be involved at any given time.

Obviously, the use of so many checking procedures added a considerable amount of overhead to the system, as far as amount of code is concerned; however, sufficient core was available, and execution speed was not significantly degraded.

V. SYSTEM DESCRIPTION

The description of the system is divided into three major areas: hardware dependencies, system characteristics, and data structures.

A. HARDWARE DEPENDENCIES

Because of the hardware equipment that was selected the following hardware dependencies exist in the current implementation of the system:

1. The system utilizes the real time clock logic as provided by the MDS system; this clock is capable of generating an interrupt of level 1, when enabled, at fixed intervals of 0.77 milliseconds. In order to be able to use this feature, a procedure named CLOCK was implemented and defined to be of type INTERRUPT 7; this allowed the PL/M 80 compiler to create the necessary code for the interrupt vector and for the routine CLOCK. A software problem needs to be explained at this point; since the development of this system was done under ISIS-II, the CLOCK procedure had to be defined as if occuring at a level other than 0, 1 or 2 (in this case 7), because ISIS-II would not allow any user generated code to be located below memory location 3000H, except for code to be used in interrupts 3 through 7, or locations 24 to 63; since the clock interrupts are of level 1 (locations 8 through 15), then in order to override this ISIS-II inconvenience, the interrupt vector generated for

CLOCK, as starting in location 56 (level 7), had to be moved to location 8 (level 1), after locating the system code in memory, and before attempting to use the real time clock; References 6, 7 and 9, describe this problem in more detail.

2. The system occupies approximately 40K bytes of physical memory for code, and approximately 12K bytes to be used for variable data; therefore a configuration of at least 52K bytes of RAM is necessary to execute the system.

3. The system utilizes an SBC 310 High Speed Mathematics Unit to perform floating point arithmetic. Although the presence of this Math Unit could be avoided by replacing its functions with appropriate software routines performing the same operations using the same formats, this is not recommended because of the excessive overhead that would result, especially with regard to execution time. Appendix D explains how the Math Unit was actually implemented.

4. The system depends in three ways on the type of terminal used: the handshaking procedure necessary to communicate between the CPU and the terminal, the code needed to control the CRT's functions, and the general features of the DATAMEDIA Elite 2500 Video Terminal - notably the programmable roll mode, the setting of privileged fields, the capability of having an addressable cursor, and the possibility of making displayed messages blink.

5. Finally, the system has a hardware dependency with

respect to the Plasma Display Unit selected, in three respects: the handshaking procedure necessary to send characters to the Plasma Display Unit; the code used to control the Plasma Display Unit functions; and the capability of the Plasma Display Unit of working either in alphanumeric or in vector mode. The SAI Plasma Display Unit has a built-in capacity to draw solid or dashed vectors by specifying the two end points defining the vector.

B. SYSTEM CHARACTERISTICS

As was established before, the system was designed to perform basically the same functions as Dead-reckoning equipment, while adding the capability of simultaneously solving maneuvering board problems. The system, as implemented in this thesis, is capable of performing the following tasks:

1. Maintain as many as 30 positions of the own ship; maintain as many as 15 positions for each contact, for as many as 15 different contacts.
2. Maintain and present a geographic plot of own ship and contacts.
3. Maintain and display a Surface Status Board.
4. Solve the following Maneuvering Board problems: (a) CPA information, and (b) course and speed of contacts.
5. At user's request, prompt for necessary inputs and update the displays, if applicable.
6. At user's request, display all the information that

exists in the system, in a pre-established format.

7. Update automatically the position of the own ship, by using its course and speed values with a frequency specified by the user.
8. Maintain a Real Time Clock.

All the above tasks are performed by the system, either automatically or at user's request. Some parameters need to be defined during system initialization; these parameters are:

1. Time zone number.
2. Local time at which the system is started.
3. Latitude and longitude values defining a selected geographical point to be used as center of a Coordinated Grid system used mainly for plotting purposes and as a reference to determine positions of the own ship and contacts.
4. Latitude and longitude values defining the starting position of the own ship.
5. Initial course value of the own ship.
6. Initial speed value of the own ship.
7. Initial scale value for the Plasma Display Unit.

The system also has two parameters with initial default values: (1) the Safe CPA Range set at 50 yards, and (2) the interval of time between updates of the own ship's positions is set at 180 seconds. Any parameter in the system can be changed at any time, with the exception of those parameters that define the boundaries of work of the system; these

fixed values are:

1. Maximum range: 100.0 miles.
2. Maximum speed: 99.9 knots.
3. Number of letters used to designate a contact: 2.
4. Minimum scale value: 00.25 miles/inch.
5. Maximum scale value: 25.00 miles/inch.
6. Zero defined for the system:

   - 0.0000009 <= "zero" <= +0.0000009 (For floating point numbers only.)
7. Minimum Safe CPA Range value: 50 yards.
8. Maximum Safe CPA Range value: 1000 yards.

Table II describes the formats used for input, internal, and output representation of the values with which the system operates, as well as the units used; Table III gives the conversion factors used in the system.

1. System Displays

As was stated previously, the system maintains two different displays: a Video Terminal presents a Surface Status Board and interactions with the user, and a Plasma Display presents a geographic plot of the positions of own ship and contacts.

a. Video Terminal Display

Figure No. 4 presents a picture of how the Video Terminal Display is arranged. The screen is divided in two areas; the upper portion of the screen consists of a fixed format presentation of information about the own ship and some contacts. From this representation, the following

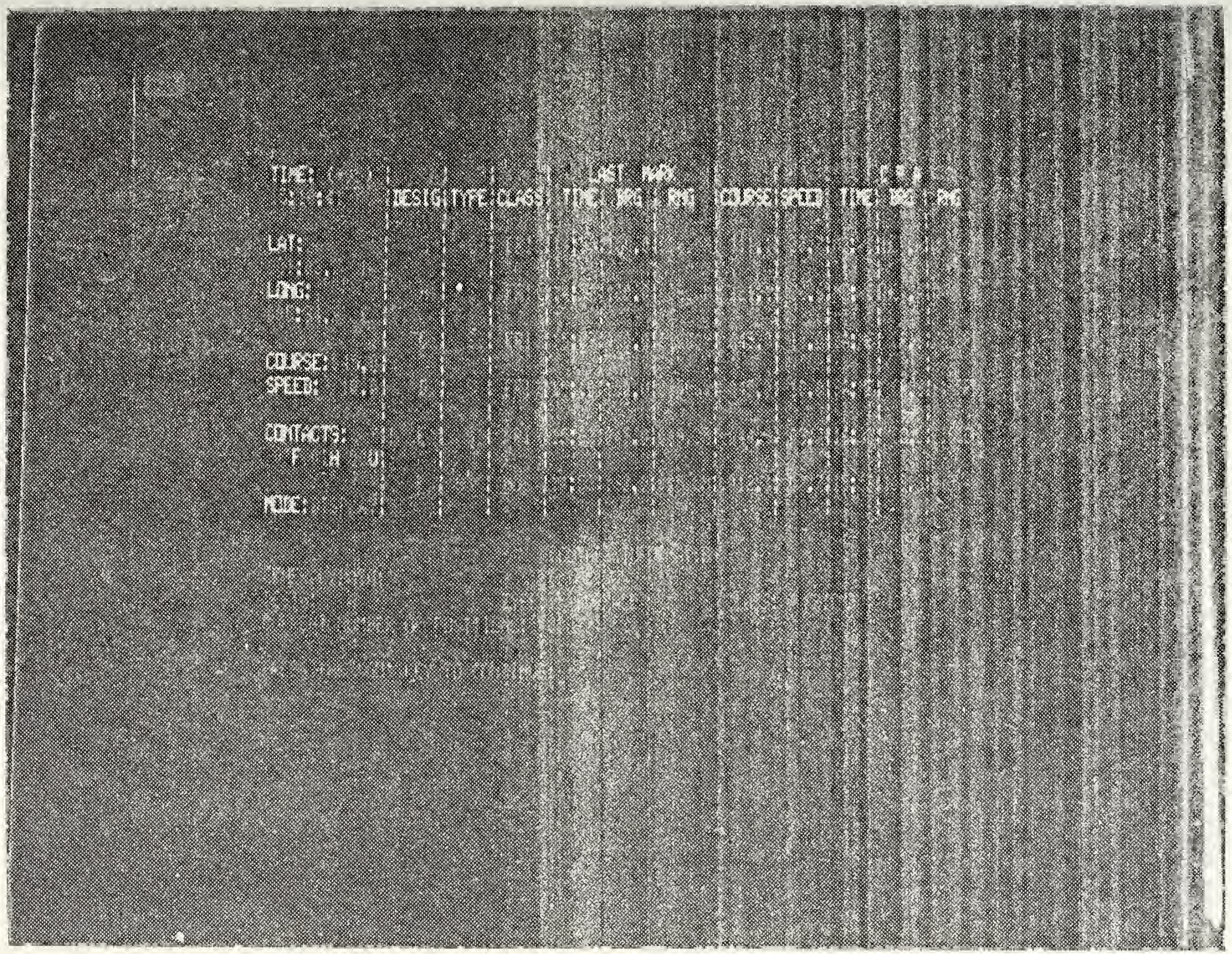
VALUE	FORMAT		UNITS	
	I/O	INTERNAL	I/O	INTERNAL
TIME	xx:xx:xx	3 bytes	hrs : min : sec	same
COURSE	xxx.x	F.P.	degrees	same
SPEED	xx.x	F.P.	knots	same
BEARING	xxx.x	F.P.	degrees	same
RANGE	xxx.x xxxxxx	F.P.	miles yards	miles
SCALE	xx.xx	F.P.	miles/inch	same
LAT.	xx:xx.x	F.P.	degrees:minutes	minutes
LONG.	xxx:xx.x	F.P.	degrees:minutes	minutes
DESIG	Aa	Address	ASCII characters	decimal value

x	Numeric character.
F.P.	Floating point representation.
A	Alphabetic character including space.
a	Alphabetic character excluding space.

TABLE II. FORMATS AND UNITS USED

1 nautical mile	2025.3716 yards.
1 degree	0.0174532925 radians.
1 minute:	0.00029089 radians.
1 minute of long. ...	1 nautical mile.
1 minute of lat.	1 nautical mile x Cos(Latitude).
PI	3.141593
1 knot	1 nautical mile / hour.

TABLE III. CONVERSION FACTORS USED



```
TIME:                                  LAST MARK                        CPA
              DESIG TYPE CLASS  TIME  BRG  RNG  COURSE SPEED  TIME  BRG  RNG

LAT:

LONG:


COURSE:
SPEED:

CONTACTS:
  F   H   U

MODE:
```

FIGURE NO. 4 VIDEO TERMINAL DISPLAY PICTURE ARRANGEMENT

information can be obtained at any time:

(1) Time. The time zone number and the local time maintained and automatically updated by the system every second.

(2) Own Ship. Latitude and longitude values indicating its last geographical position as determined automatically by the system at least every T time, where T is a period of time as selected by the user. Information about the course and speed is also available.

(3) System. Information about the total number of contacts in the system, classified by their corresponding class: Friendly (F), Hostile (H), and Unknown (U). Information about the mode in which the system is operating is also displayed. This will be explained in the following paragraphs.

(4) Contacts. Complete information about six contacts is displayed. The following items are provided for each contact: (a) designation, (b) type: Surface (S) or Sub-surface (SS), (c) class: Friendly (FRI), Hostile (HOS), or Unknown (UNK), (d) last mark: time, bearing and range, (e) course, if known, (f) speed, if known, (g) CPA information, if known: time, bearing and range, or one of the following three possible messages: "timeCOLLISION", "SAME CRS & SPD", and "MOVING AWAY".

It should be noted that, although the system is capable of maintaining up to 15 contacts, information about only six of them will be constantly present at the display;

the user has the capability of selecting which contacts he desires to be displayed in this way, or as will be explained later, he can also obtain information about any contact (temporarily) in the lower portion of the screen.

The lower portion of the screen consists of the last eight rows and is used during Input and Display operations; it has no fixed format and if the system is not executing an Input or Display operation, it contains only the prompt ("%") symbol displayed in its upper left corner.

As mentioned previously, the system operates in three different modes; these modes are: (1) Initialization, (2) Input, and (3) Display. The Initialization mode is required only once at the beginning of the execution of the system, and it is indispensable for the operation of the system, as was explained previously; it can not be requested by the user once the system is operating. The Input and Display modes are determined by the way the system is performing at any given time. The system operates in the Input mode by default. Both modes require interaction with the user.

(1) Input Mode. The system is operating in Input mode any time an Input operation is being performed. The following Input operations can be requested by the user:

(a) Modify Coordinate Grid Origin parameters; optional: latitude and longitude.

(b) Modify Own Ship parameters; optional: latitude, longitude, course and speed.

(c) Create a Contact; required: signation, type, class, bearing, and range; optional: urse and speed.

(d) Remove a contact; required: signation.

(e) Redesignate a Contact; required: old d new designations.

(f) Update a Contact; required: signation; optional: type, class, bearing, range, course d speed.

(g) Contacts to Display: select which ntacts are desired to be displayed permanently in the rface Status Board.

(h) Time; optional: time zone value, system ock value, and time between updates.

(i) Safe CPA Range: modify the Safe CPA nge parameter.

(j) Wind: enter/modify wind parameters; quired: direction and speed.

(k) Scale: modify the graphics scale value.

(l) Plasma Reorientation: reorient the cture displayed at the Plasma Display Unit.

All these operations are performed using e lower portion of the screen and are divided into various ases ("pages") in order to allow a better utilization of e screen; any Input operation can be requested by pressing e appropriate key at the keyboard.

(2) Display Mode. The system is operating in Display mode any time a Display operation is being performed. The following Display operations can be requested by the user:

(a) Origin: display information about the Coordinate Grid Origin parameters (latitude and longitude).

(b) Scale: displays information about the Graphics Scale currently used by the system.

(c) Own Ship: displays information about the following own ship parameters: latitude, longitude, X and Y values in the user defined Coordinate Grid System being used, course and speed.

(d) Contact Information: displays information about a specified contact's parameters; requires: designation; provides: type, class, number of positions maintained by the system, latitude, longitude, X and Y values in the user defined Coordinate Grid System being used, last mark's time, bearing and range, and if possible, gives information about course, speed, CPA parameters, and estimated actual position.

(e) Contacts in System: displays information about the designations of all the contacts in the system, if any.

(f) Safe CPA Range: displays the value of the Safe CPA Range parameter.

(g) Wind: displays information about the wind.

(h) Display Update Time: displays the value of the current Time Between Updates being used.

All these operations are performed using the lower portion of the screen, and are divided into various phases ("pages") in order to allow a better utilization of the screen. Any Display operation can be requested by pressing the appropriate key at the keyboard.

Figure No. 4 presents a view of how the Video Terminal display looks during system operation; Figure No. 5 presents a layout of the keyboard used and Figure No. 5a the arrangement of the various Input and Display keys.

(3) Other Function-Defined Keys. The keyboard has also two other special function keys:

(a) Rubout Key. Used to backspace the cursor when inputting information into the system.

(b) "GO" key. Used to advance the various "pages" in which the Display operations are divided.

b. Plasma Display

The Plasma Display is used to present a geographical plot of the own ship and contact positions. It displays the geographical picture that is defined through a system-defined and user-controllable "window"; this window is used to focus on the geographical area that is of interest to the user. Two mechanisms control the "window":

(1) Scale. This defines the scale at which the plotting is desired to be presented, determining the size of the window.

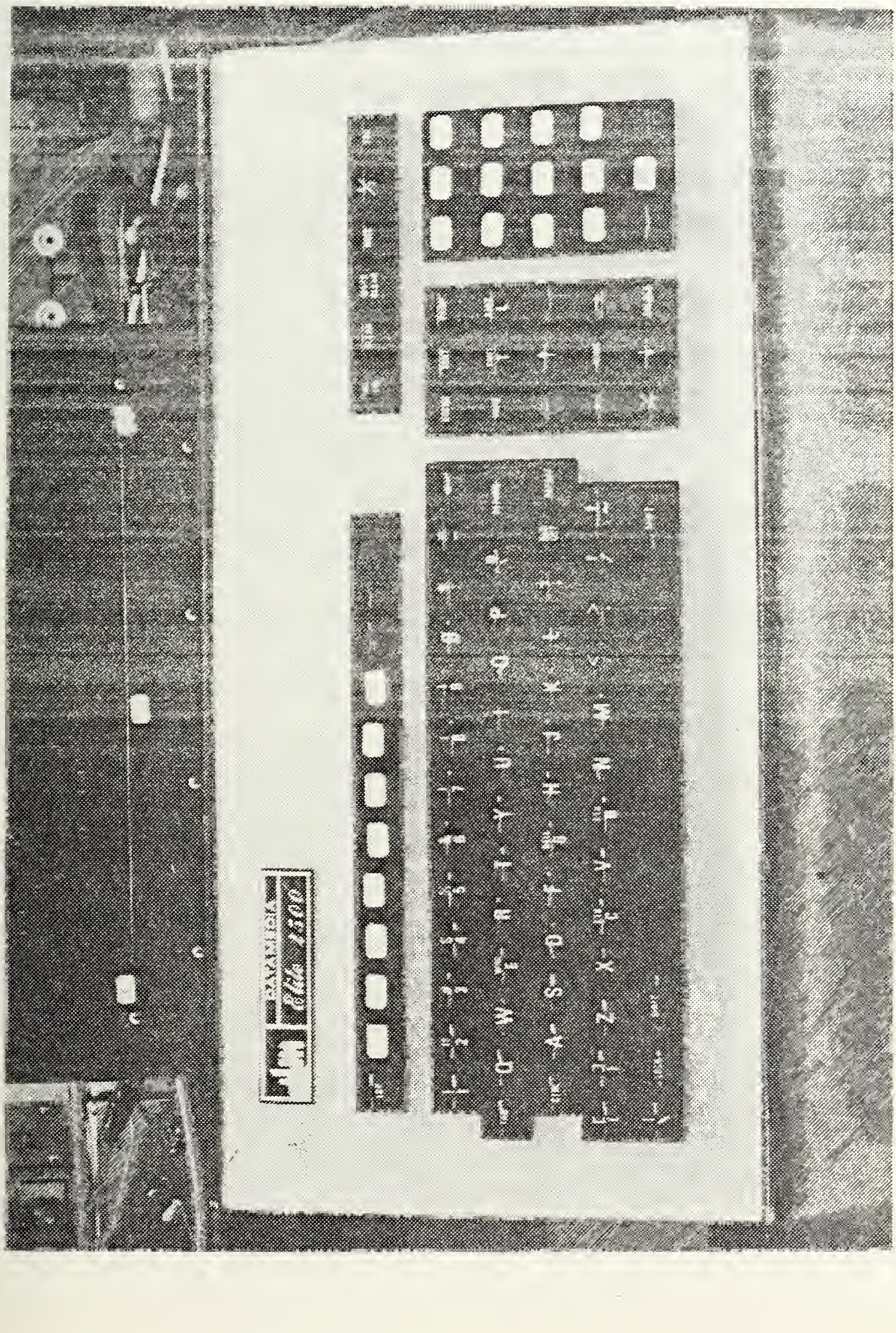


FIGURE NO. 5 KEYBOARD LAYOUT

DISPLAY KEYS

1	2	3	4	5	6	7	8

1. Origin.
2. Scale.
3. Own Ship.
4. Contact Information.
5. Contacts in System.
6. Safe CPA Range.
7. Wind.
8. Display Update Time.

INPUT KEYS

1	2	3
4	5	6
7	8	9
10	11	12
	13	

1. Create a Contact.
2. Modify Own Ship.
3. Modify Coordinate Grid Origin.
4. Remove a Contact.
5. Contacts to Display.
6. Wind.
7. Redesignate a Contact.
8. Time.
9. Scale.
10. Update a Contact.
11. Safe CPA Range.
12. Plasma Reorientation.
13. GO key.

FIGURE NO. 5a INPUT AND DISPLAY KEYS ARRANGEMENT

(2) Picture Reorientation. This specifies the window's center. Three different methods were used: (a) fixed reorientation, by making as new center any one of eight (8) pre-defined points in the picture, (b) by making the last position of the own ship to be the new center, and (c) by making the last position of any contact to be the new center of the picture.

The Plasma Display also presents the current value of the scale being used in its upper left corner. The positions of the own ship are marked with bright circles connected by solid vectors, while the positions of the contacts are connected by dashed vectors and marked with two different symbols: a cross if the contact is Hostile or Unknown, and a circle if the contact is Friendly; the designation of any contact plotted at the Plasma Display is presented close to its first plotted position.

Figure No. 6 presents a view of how the Plasma Display appears during system operation; Appendix A gives the algorithms used for establishing the window and for forming the picture to be presented.

C. DATA STRUCTURES

The main data structures used in the system can be classified in three categories: (1) data structure used to represent system parameters, (2) data structures used to represent the own ship information, and (3) data structures used to represent contacts information. The main type of data structure used was the STRUCTURE, as provided

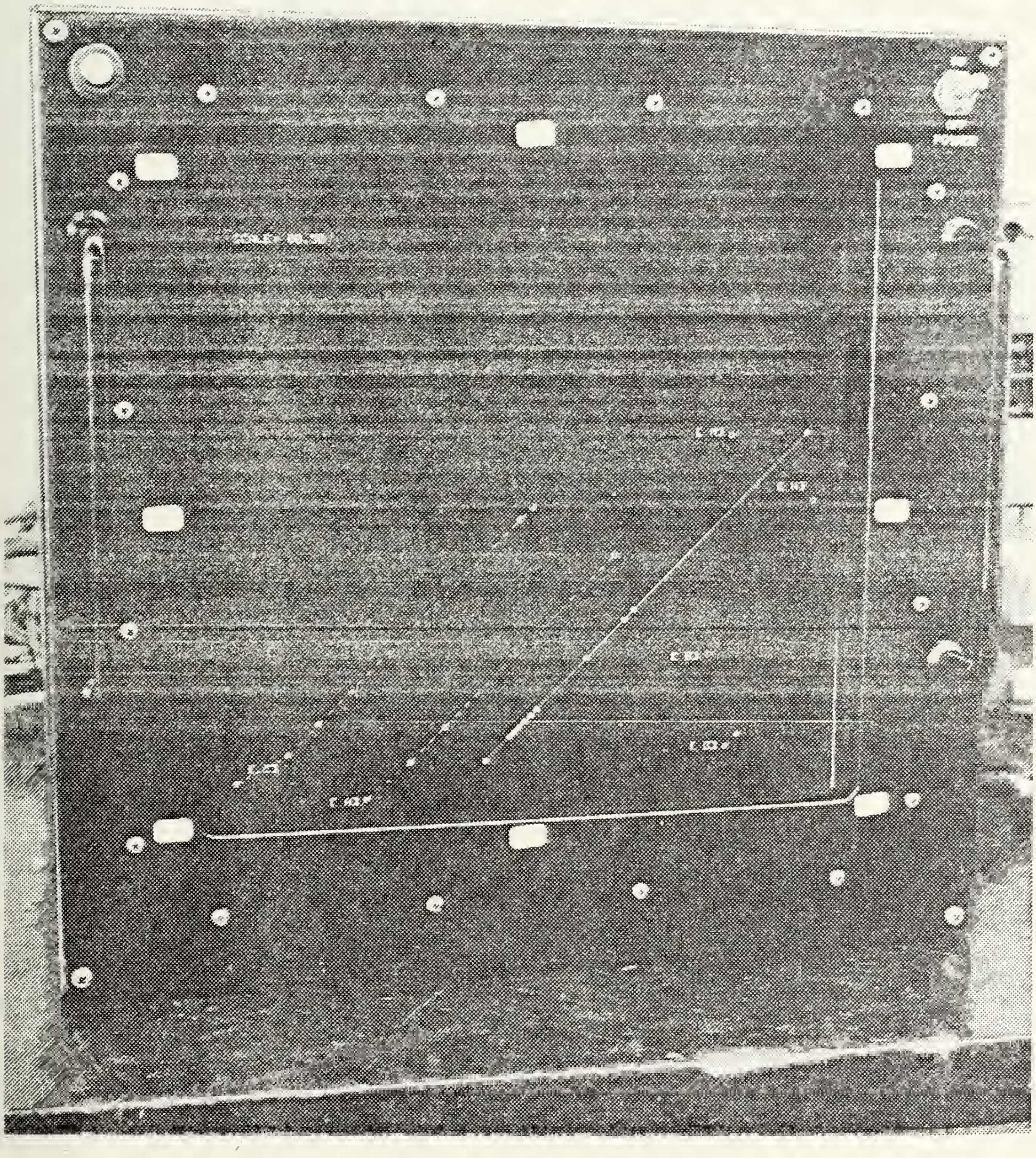


FIGURE NO. 6 PLASMA DISPLAY VIEW

by PL/M 80 [Ref. 9], which basically allows the programmer by using one identifier, to refer to a collection of STRUCTURE MEMBERS which may have different types, such as ARRAY, ADDRESS variables and BYTE variables. In PL/M 80, a BYTE variable is an 8-bit value occupying a single byte of storage, an ADDRESS variable is a 16-bit value occupying two consecutive bytes of storage, and an ARRAY is a vector composed of BYTE or ADDRESS variables. The most sophisticated data structure that PL/M 80 supports directly is the Array of Structures with Arrays inside the structures. With the capability that PL/M 80 has of allowing a variable reference to be either fully qualified, partially qualified, unqualified (references to entire arrays or structures), or by using pointers and indirect references, it was possible to simulate other more complex data structures such as circular linked lists and plexes, as will be explained later.

1. Data Structure for the System Parameters

Figure No. 7 describes the data structure used to represent most of the system parameters; as can be observed, it is a structure composed of eight (8) members with the following description:

a. LAT: array of 4 bytes used to represent the floating point value of the latitude parameter used to define the Coordinate Grid Origin.

b. LONG: array of 4 bytes used to represent the floating point value of the longitude parameter used to

```
                SYSTEM

        --------------------
        :                  :
        :     LAT   (4)    :
        :                  :
        :------------------:
        :                  :
        :     LONG  (4)    :
        :                  :
        :------------------:
        :                  :
        :    SCALE   (4)   :
        :                  :
        :------------------:
        :                  :
        :  WIND$DIR  (4)   :
        :                  :
        :------------------:
        :                  :
        :  WIND$SPD  (4)   :
        :                  :
        :------------------:
        :                  :
        :  NUM$ZONE  (5)   :
        :                  :
        :------------------:
        :                  :
        : CONTACT$KIND (3) :
        :                  :
        :------------------:
        :                  :
        :   NUMCTS  (1)    :
        :                  :
        :------------------:
```

FIGURE NO. 7 DATA STRUCTURE FOR THE SYSTEM PARAMETERS

define the Coordinate Grid Origin.

c. SCALE: array of 4 bytes used to represent the floating point value of the graphics scale parameter.

d. WIND$DIR: array of 4 bytes used to represent the floating point value of the wind direction.

e. WIND$SPD: array of 4 bytes used to represent the floating point value of the wind speed.

f. NUM$ZONE: array of 5 bytes used to represent the ASCII characters defining the value of the Time Zone number.

g. CONTACT$KIND: array of 3 bytes used to represent the total number of contacts in each class.

h. NUMCTS: byte variable used to represent the number of contacts at any time in the system.

It should be noticed that this data structure is designed only to maintain the parameters described at any moment; a log of changes or modifications to the parameters described is not kept.

2. Data Structures for Own Ship Parameters

Figure No. 8 describes the data structures used to represent the own ship parameters; the OWN$SHIP$INFO structure is used to maintain a set of parameters for which no log of changes or modifications is maintained, and also to allow indirect access to the OWN$SHIP data structure. Its four (4) members have the following description:

a. LAT: array of 4 bytes used to represent the

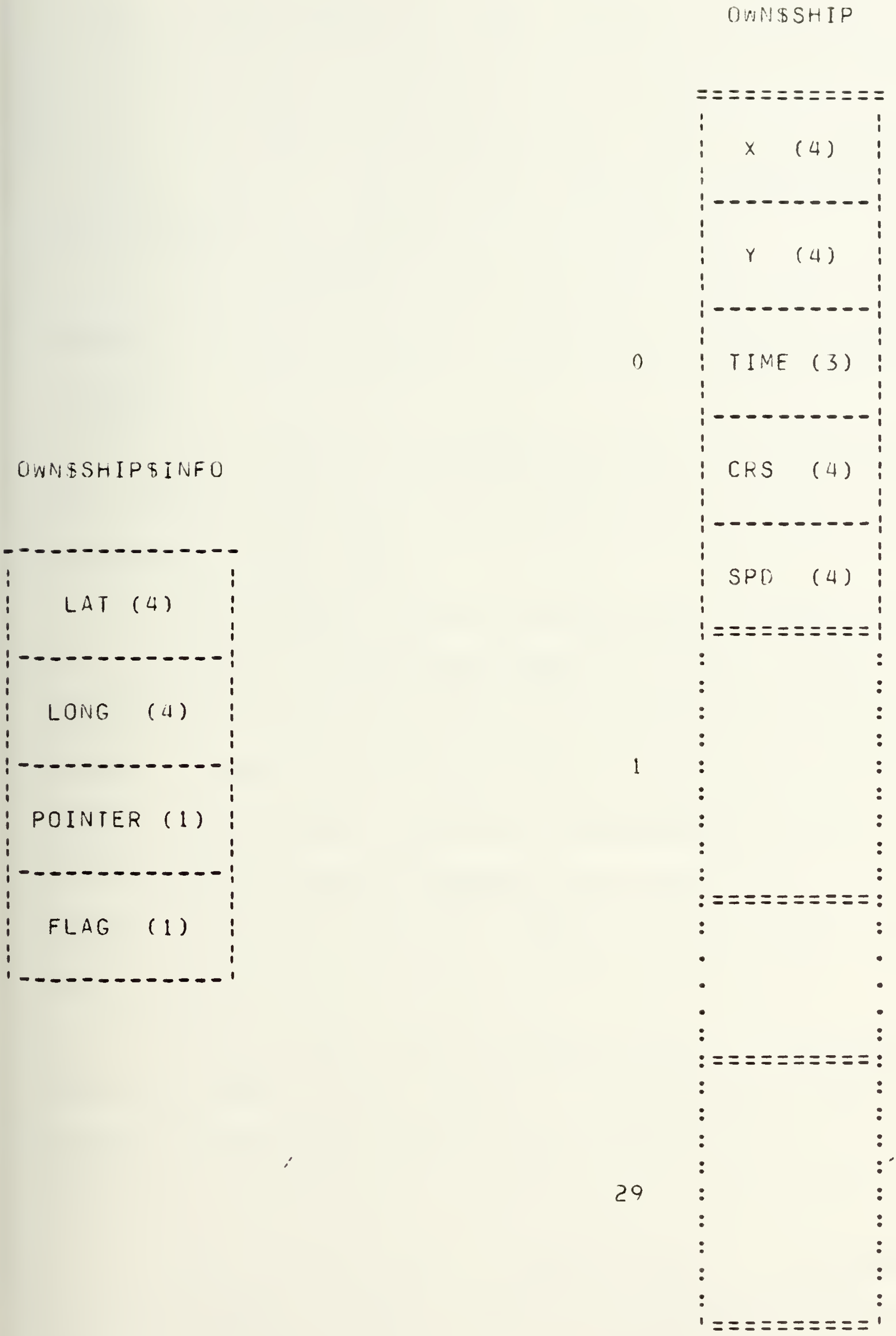


FIGURE NO. 8 DATA STRUCTURES FOR OWN SHIP PARAMETERS

floating point value of the latitude parameter defining the geographical position of the own ship.

b. LONG: array of 4 bytes used to represent the floating point value of the longitude parameter defining the geographical position of the own ship.

c. POINTER: byte variable used to access the OWN$SHIP data structure, by defining its most recently used member.

d. FLAG: byte variable used to indicate whether or not the 30 members of the OWN$SHIP data structure have been accessed at least once.

The OWN$SHIP Array of Structures is used to maintain up to thirty (30) different sets of values for the own ship parameters, thus maintaining a log of the 30 (or fewer) most recent values of the own ship parameters. By using the members POINTER and FLAG of the OWN$SHIP$INFO structure, it is possible to use the OWN$SHIP data structure as a circular queue, with each of its 30 members having the following description:

a. X: array of 4 bytes used to represent the floating point value of the X parameter defining the position of the own ship in the Coordinated Grid System defined.

b. Y: array of 4 bytes used to represent the floating point value of the Y parameter defining the position of the own ship in the Coordinated Grid System defined.

c. TIME: array of 3 bytes used to represent the real time clock value (hours, minutes and seconds) at which the corresponding member was current.

d. CRS: array of 4 bytes used to represent the floating point value of the course parameter.

e. SPD: array of 4 bytes used to represent the floating point value of the speed parameter.

Notice that each member of the OWN$SHIP data structure is capable of providing enough information to locate the own ship; up to 30 locations and time may be recorded.

3. Data Structures for the Contact Parameters

Figure No. 9 describes the data structures used to represent contact parameters. The CONTACT$INFO Array of Structures, composed of fifteen (15) members, was used to represent a set of parameters for up to 15 different contacts, and was also used to allow indirect access to the CONTACT$POSI data structure; each member of CONTACT$INFO represented a different contact and was used to access 15 corresponding members in the CONTACT$POSI data structure, as will be described. Each member of the CONTACT$INFO data structure had the following configuration:

a. DESIG: address value used to represent the decimal value of the designation used to refer to a contact.

b. TYPE: byte variable used to represent the type of the contact.

c. KIND: byte variable used to represent the class

```
       CONTACT$INFO                            CONTACT$POSI

   ==================                          ============
   !    DESIG (2)     !                         !  X   (4)  !
   !------------------!                         !-----------!
   !    TYPE   (1)    !                         !  Y   (4)  !
   !------------------!                         !-----------!
   !    KIND   (1)    !                         ! TIME (3)  !
   !------------------!                         !-----------!
   !  CRS$FLAG (1)    !                      0  ! CRS  (4)  !
 0 !------------------!                         !-----------!
   !  SPD$FLAG (1)    !                         ! SPD  (4)  !
   !------------------!                         !-----------!
   ! OS$POINTER (1)   !                         ! BRG  (4)  !
   !------------------!                         !-----------!
   !  POINTER   (1)   !                         ! RNG  (4)  !
   !------------------!                         '==========='
   !    FLAG (1)      !                         :           :
   '=================='                         :           :
   :                  :                      1  :           :
   :                  :                         :           :
 1 :                  :                         :           :
   :                  :                         :===========:
   :                  :                         :           :
   :==================:                         .           .
   :                  :                         .           .
   .                  .                         .           .
   .                  .                         .           .
   .                  .                         .           .
   :                  :                         :           :
   :==================:                         :===========:
   :                  :                         :           :
   :                  :                         :           :
14 :                  :                    224  :           :
   :                  :                         :           :
   :                  :                         :           :
   '=================='                         '==========='
```

FIGURE NO. 9 DATA STRUCTURES FOR CONTACT PARAMETERS

of the contact.

d. CRS$FLAG: byte variable used to indicate whether or not there was information about the course of the contact.

e. SPD$FLAG: byte variable used to indicate whether or not there was information about the speed of the contact.

f. OS$POINTER: byte variable used to indicate which of the 15 corresponding members of the CONTACT$POSI data structure was current when the own ship made its last change in course or speed; it is mainly used when solving Maneuvering Board problems.

g. POINTER: byte variable used to access the 15 corresponding members of the CONTACT$POSI data structure, by defining its most recently used member.

h. FLAG: byte variable used to indicate whether or not the 15 corresponding members of the CONTACT$POSI data structure have been accessed at least once.

The CONTACT$POSI Array of Structures is used to maintain up to 15 different sets of values for the parameters of up to 15 different contacts, thus maintaining a log of the 15 (or fewer) most recent values for the parameters of each contact. Its 225 members are accessed indirectly by the POINTER element in each member of CONTACT$INFO, according to the following method:

Relative position of member in CONTACT$INFO: N.

Members in CONTACT$POSI to which N is allowed

to access: 15 x N up to (15 x (N + 1)) - 1

Thus, for example, the contact defined as member 0 in CONTACT$INFO, has its 15 sets of parameters represented in members 0 thru 14 of CONTACT$POSI, and the contact defined as member 8 in CONTACT$INFO, has its 15 sets of parameters represented in members 120 thru 134 of CONTACT$POSI.

Also, the 15 members of CONTACT$POSI that are allocated for each contact defined in CONTACT$INFO, are used as a circular queue, in a similar way to that described for the own ship.

Each of the 225 members has the following configuration:

a. X: array of 4 bytes used to represent the floating point value of the X parameter defining the position of the contact in the Coordinated Grid System defined.

b. Y: array of 4 bytes used to represent the floating point value of the Y parameter defining the position of the contact in the Coordinated Grid System defined.

c. TIME: array of 3 bytes used to represent the real time clock value (hours, minutes and seconds) at which the corresponding member was current.

d. CRS: array of 4 bytes used to represent the floating point value of the course parameter.

e. SPD: array of 4 bytes used to represent the

oating point value of the speed parameter.

f. BRG: array of 4 bytes used to represent the oating point value of the bearing parameter.

g. RNG: array of 4 bytes used to represent the oating point value of the range parameter.

VI CONCLUSION

RESULTS

Presently, the system will handle all functions rrently provided by the DRT plotter, while at the same me solving some Maneuvering Board problems: CPA formation and determination of course and speed of ntacts.

Since all the inputs to the system are done manually, an tensive control program was developed in order to make it ry difficult for the user to enter erroneous data, oviding up to six levels of correctness.

Information about the own ship and any one of 15 fferent contacts can be displayed by pressing one nction-defined key for each case. Also, the geographical ot displayed at the Plasma Unit can be reoriented and dified in scale by using two other function-defined keys.

The displays provided by the system can easily be plicated, thus allowing the possibility of having all the formation displayed in remote stations where needed.

FUTURE WORK

The system can be a valuable tool aboard non-NTDS ships implemented. Several extensions are possible, however, in th hardware and software.

The system can be modified in order to accept input data rectly from the own ship's pitometer and gyrocompass, and

from the radar and sonar sensors, in order to reduce time required from the operator to enter the same data, improving the system's performance.

The system could also be modified in order to allow the of Magnetic Bubble Memory as mass storage media, to tain a log of all the necessary data to be able to nstruct events, and to start up the system again in case loss of power or other failure.

Also, the capability of solving other Maneuvering Board lems such as interception, scouting and torpedo firing, d easily be added.

The system could also be implemented by using several le Board Computers working in parallel, with each one ing one or more specific problems. Also, the remote ions could be provided with intelligent terminals with capability of displaying the information necessary, out disturbing the computer's processing.

The results of this thesis indicate that microcomputer nology can feasibly be applied to the ideas above and to r similar problems.

A microprocessor-based system as the one described, has potential to provide any ship with a digital surface-surface contact plotting capability, without the expense NTDS size equipment. It can reduce the manning of rway watches while improving the quality of the tactical rmation available to the Officer of the Deck.

APPENDIX A ALGORITHM DESCRIPTION

A THE CLOSEST POINT OF APPROACH (CPA)

1. The Basic Relative Movement Problem

K. H. Kerns and R. S. Cooper [Ref. 3] have described a way of solving Maneuvering Board problems with the aid of a microcomputer. As described in that reference, aneuvering Board problems are divided in two basic ategories.

One is the relative plot where the CPA of contacts eing tracked can be calculated. The center of the plot epresents the "reference" or "own" ship and any other point epresents the position of a "maneuvering" ship, plotted in rue bearing and range from the own ship at various times.

The other category is the vector diagram or the 'triangle of courses and speeds"; this allows the operator to calculate the course and speed of any maneuvering ship (a contact) given the own ship course and speed, and relative course and speed of the contact (obtained from the relative plot). Figure No. 10 shows how the two categories are applied and how they interact one with another.

2. Other Uses of the Maneuvering Board

All Maneuvering Board problems utilize the basic relative motion problem discussed above; in addition to determining CPA information, course, and speed of a contact, the Maneuvering Board can also be used to find the required course and speed to take station on or to intercept another

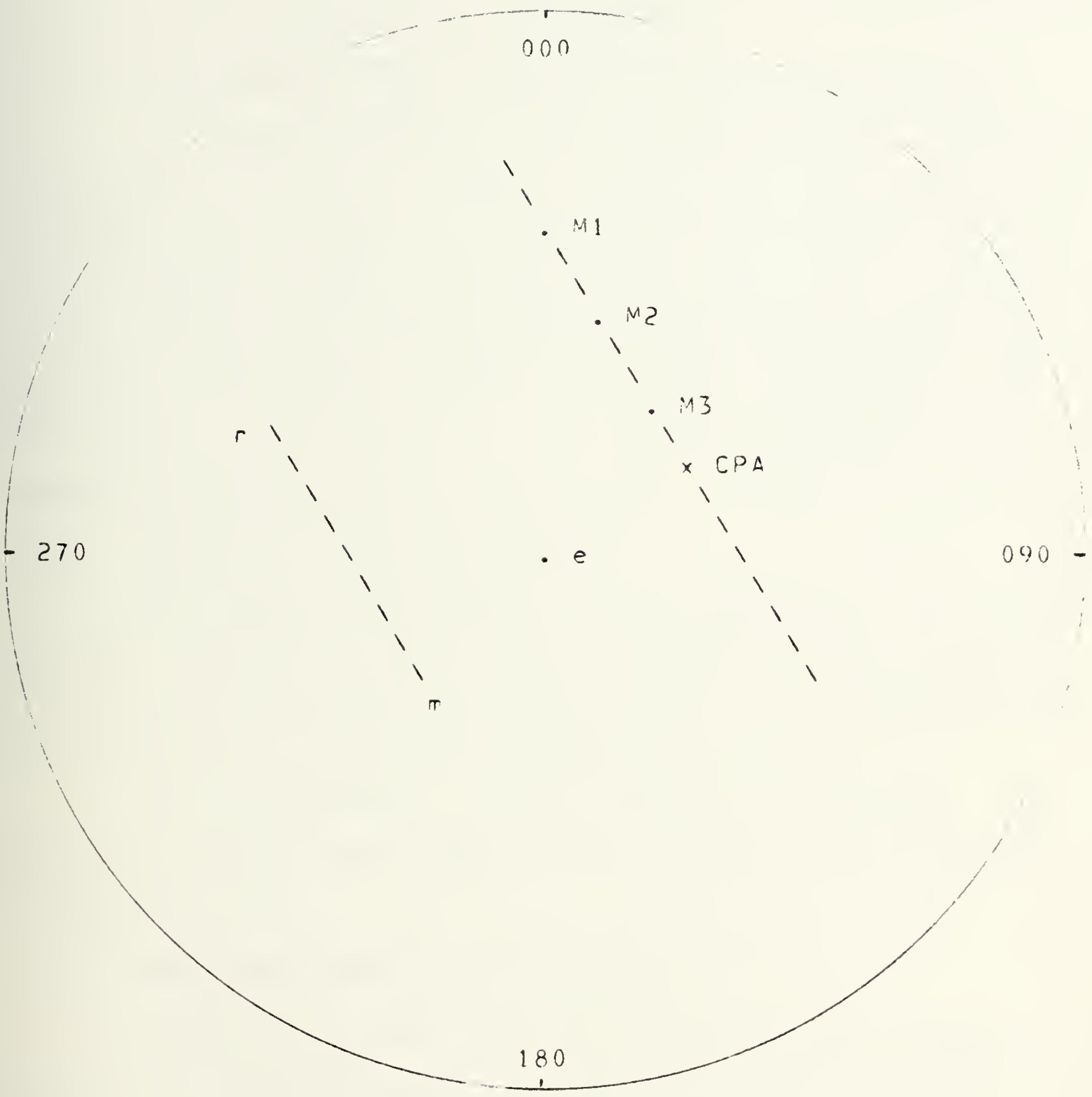


FIGURE NO. 10 MANEUVFRING BOARD SCHEMATIC

ship, to find true wind, or to find courses and speeds for scouting and torpedo-firing situations.

3. Maneuvering Board problems solutions implemented

As described before, the system was mainly designed to provide a graphical display of the own ship and contacts being tracked; secondarily, the system provides a surface status display of information about those contacts and about own ship. Only the CPA, course, and speed of contacts are calculated automatically; the other functions of the Maneuvering Board are not duplicated.

4. Problems that Normally Occur

In a Maneuvering Board problem solution, all the data are recorded manually by the plotter; these data are provided aurally by the radar operator.

This interaction is somewhat error-prone; the positions plotted often appear scattered like the ones shown in Figure No. 11.

5. The Least-squares Fit Approach

In order to smooth the data utilized and to obtain a straight line representing the line of relative motion for a certain number of plotted positions, the least-square fit method was chosen.

The approach employed requires two to five contact positions. A least-square fit is used to determine the slope and the Y axis intercept. Once these parameters are obtained the CPA, course, and speed calculations are performed in a straightforward way. According to Reference No. 15 the

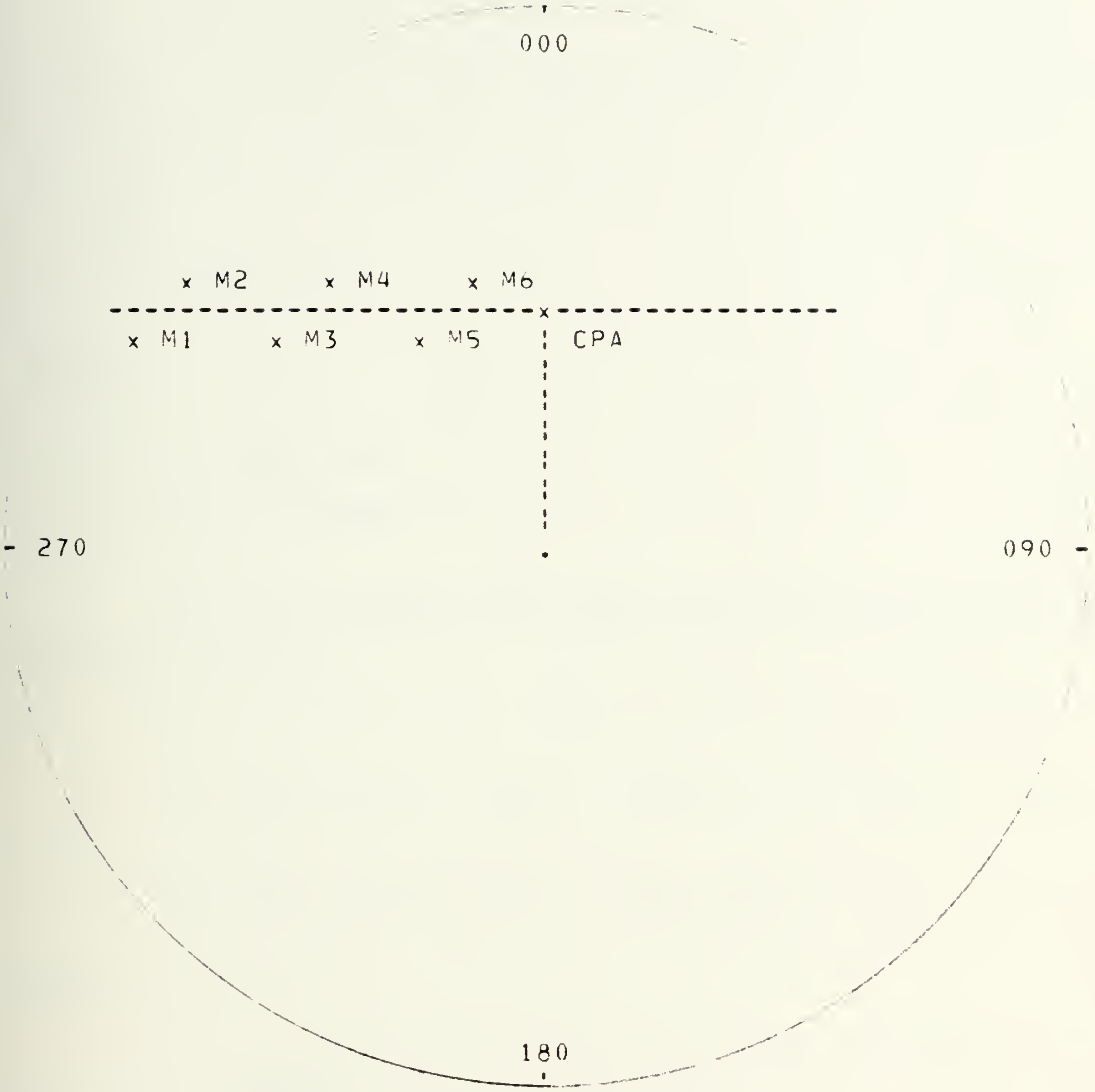


FIGURE NO. 11 USUAL PLOTTING AT THE MANEUVERING BOARD

method is as follows:

Let N be the number of positions of a contact obtained from the radar repeater/sonar (the value of N in the system has as bounds, 2 and 5). The equation of a straight line can be represented by:

$$y = MX + B$$

where:

M = slope, and

B = Y intercept.

The least-square fit method gives the solution for M and B as follows:

$$M = (s(0) \times t(1) - s(1) \times t(0))/(s(0) \times s(2) - s(1)^2)$$

$$B = (s(2) \times t(0) - s(1) \times t(1))/(s(0) \times s(2) - s(1)^2)$$

where:

$$s(k) = \sum_{i=0}^{N-1} X(i)^k, \quad k = 0,1,2$$

$$t(k) = \sum_{i=0}^{N-1} Y(i) \times X(i)^k, \quad k = 0,1$$

6. CPA Algorithm

Certain combinations of data require special treatment in CPA, course, and speed calculations. These special cases are shown in Figure No. 12.

For case 0 the contact has the same course and speed as the own ship; then the CPA can not be calculated, because the contact is permanently at CPA.

For case 1 the contact has as direction of relative motion (relative course) the values of 000 or 180 degrees and thus the slope of the relative motion line will have an infinite value.

For case 2 the contact has as direction of relative motion the values of 090 or 270 degrees and thus the slope of the relative motion line will have the value of 0.

There is a fourth case where the contact is on a collision course with the own ship.

In the general case, the sequence of calculations is as follows:

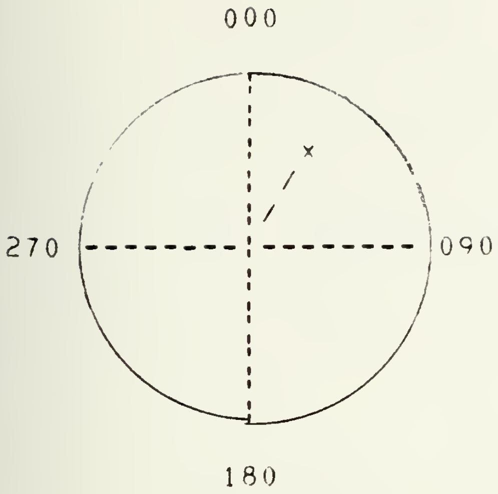
a. Take at least 2 marks of a contact (time, bearing, and distance).

b. Convert bearing and distance to relative values of x and y:

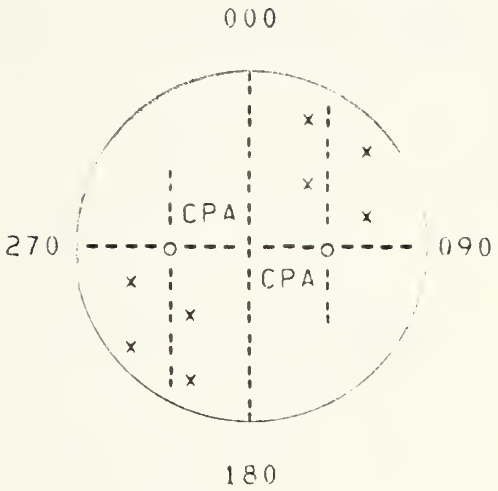
REL$X = RNG x sin(BRG)

REL$Y = RNG x cos(BRG)

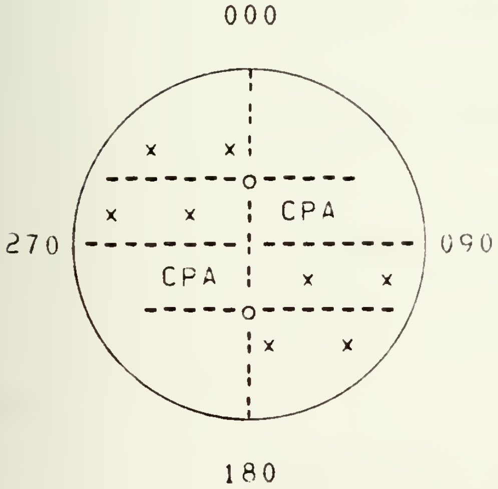
c. Compute slope of smoothed relative motion line:



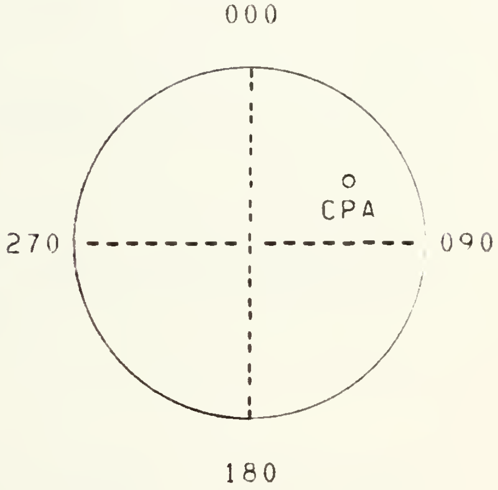
CASE 0



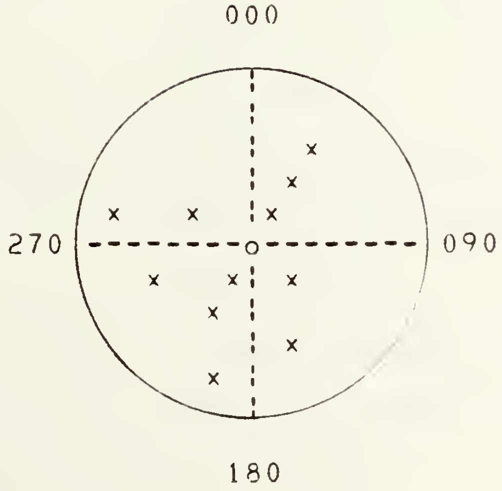
CASE 1



CASE 2



CASE 3



CASE 4

FIGURE NO. 12 SPECIAL CASES

$$M = (s(0) \times t(1) - s(1) \times t(0))/(s(0) \times s(2) - s(1)^2)$$

d. Compute Y intercept:

$$\mathrm{Y\$CUT} = (s(2) \times t(0) - s(1) \times t(1))/(s(0) \times s(2) - s(1)^2)$$

e. Compute relative course:

$$\mathrm{Y1} = M \times \mathrm{REL\$X}(1) + \mathrm{Y\$CUT}$$

$$\mathrm{Y2} = M \times \mathrm{REL\$X}(2) + \mathrm{Y\$CUT}$$

$$\mathrm{REL\$CRS} = \tan^{-1}((\mathrm{REL\$X}(2) - \mathrm{REL\$X}(1))/(\mathrm{Y2} - \mathrm{Y1}))$$

where:

REL\$X(2) - obtained from the last position used for the Least-square Fit calculation

REL\$X(1) - obtained from the first position used for the Least-square Fit calculation

REL\$Y(2) - similar to REL\$X(2)

REL\$Y(1) - similar to REL\$X(1)

f. Compute relative speed:

$$\mathrm{DELTA\$X} = \mathrm{REL\$X}(2) - \mathrm{REL\$X}(1)$$

$$DELTA\$Y = REL\$Y(2) - REL\$Y(1)$$

$$DELTA\$T = TIME(2) - TIME(1)$$

$$REL\$SPD = \frac{\sqrt{DELTA\$X^2 + DELTA\$Y^2}}{DELTA\$T}$$

where:

TIME(1) - local time at which the last position of a contact used in the least-square fit is entered into the system.

TIME(2) - local time at which the first position of a contact used in the least-square fit is entered into the system.

g. Compute true course and speed of a contact:

Given own ship's course (CO), own ship's speed (SO), speed of relative motion (REL$SPD), and the relative motion course (REL$CRS), then:

$$X1 = SO \times \sin(CO)$$

$$Y1 = SO \times \cos(CO)$$

$$X2 = REL\$SPD \times \sin(REL\$CRS)$$

$$Y2 = REL\$SPD \times \cos(REL\$CRS)$$

Thus, the X/Y components of the maneuvering ship's vector are X1 + X2 and Y1 + Y2 where the maneuvering ship's speed is:

$$\sqrt{(X1 + X2)^2 + (Y1 + Y2)^2}$$

and the course is:

$$\tan^{-1}((X1 + X2)/(Y1 + Y2))$$

h. Compute CPA

$$X\$CPA = (M \times (M \times REL\$X(1) - Y1))/(M^2 + 1)$$

$$Y\$CPA = (Y1 - M \times X1)/(M^2 + 1)$$

$$CPA\$TIME = \frac{\sqrt{(X\$CPA - REL\$X(1))^2 + (Y\$CPA - Y1)^2}}{REL\$SPD} + TIME(1)$$

$$CPA\$RANGE = \sqrt{X\$CPA^2 + Y\$CPA^2}$$

$$CPA\$BEARING = \tan^{-1}(X\$CPA / Y\$CPA)$$

NOTE: Y1 is the value calculated at the item e. above, not the one at g.

As a final comment, the case 4 shown in Figure No. 3 represents the situation when a contact is in collision with the own ship. It is easily seen that a contact in collision has the bearings of the various positions with approximately the same value while the range is reducing. The system was

designed with a safe CPA range value (SAFE$RNG) as parameter which can be changed from 50 yards (default value) up to 1000 yards; thus, any CPA range below that parameter value will set the information about the CPA of a given contact as being in collision with the own ship.

Beyond that, the CPA algorithm checks for a contact that already passed its CPA and a message "MOVING AWAY" is issued.

B. GRAPHICS ON PLASMA DISPLAY

1. Physical Considerations

In order to provide elements for designing the algorithm for interacting with the plasma display some physical parameters for the AN/UYQ-10, Plasma Display set [Ref. 13] had to be taken into account:

a. Panel parameters:

Active area : 8.55" x 8.55"

Addressable matrix : 512 x 512

Dot spacing : 0.0167" center-to-center, 60 per inch

Light spot size : 10 to 12 mils

b. Character size:

5 x 7 matrix : 80 x 120 mils

2. Plasma Display Unit Capabilities

Such capabilities included the capability to:

a. Set status of the Plasma Unit (busy or not)

b. Clear Plasma panel

c. Clear vectors

d. Receive X/Y coordinates from CPU

e. Set alphanumeric mode

f. Set vector mode (solid or dashed vector capability)

3. Algorithm Design

Figure No. 13 shows how the Plasma Panel was set in he coordinate system; the points 1, 3, 5, and 7 delimit the rea where the Plasma Panel is located. The point 7 epresents the origin of the Plasma Panel (ORIGIN$X, RIGIN$Y).

In order to allow the display of all the information ecessary to the Plasma Unit and generated by the system, he "PLASMA$MODULE" module was designed.

a. Windowing

The windowing process was developed as a ransformation process which enables the Plasma Panel to over a region in the coordinate grid system.

The scale set by the operator as an initial arameter controls the windowing process and it can vary rom .25 miles/inch up to 25.00 miles/inch.

In Figure No. 13 the name WINDOW marks the size f a square representing the region covered by the Plasma anel; this value was obtained by setting:

WINDOW = SCALE x 8.55

b. Procedure to check if a given position falls ithin the limits of the defined "window"

A mechanism was implemented to check if it was

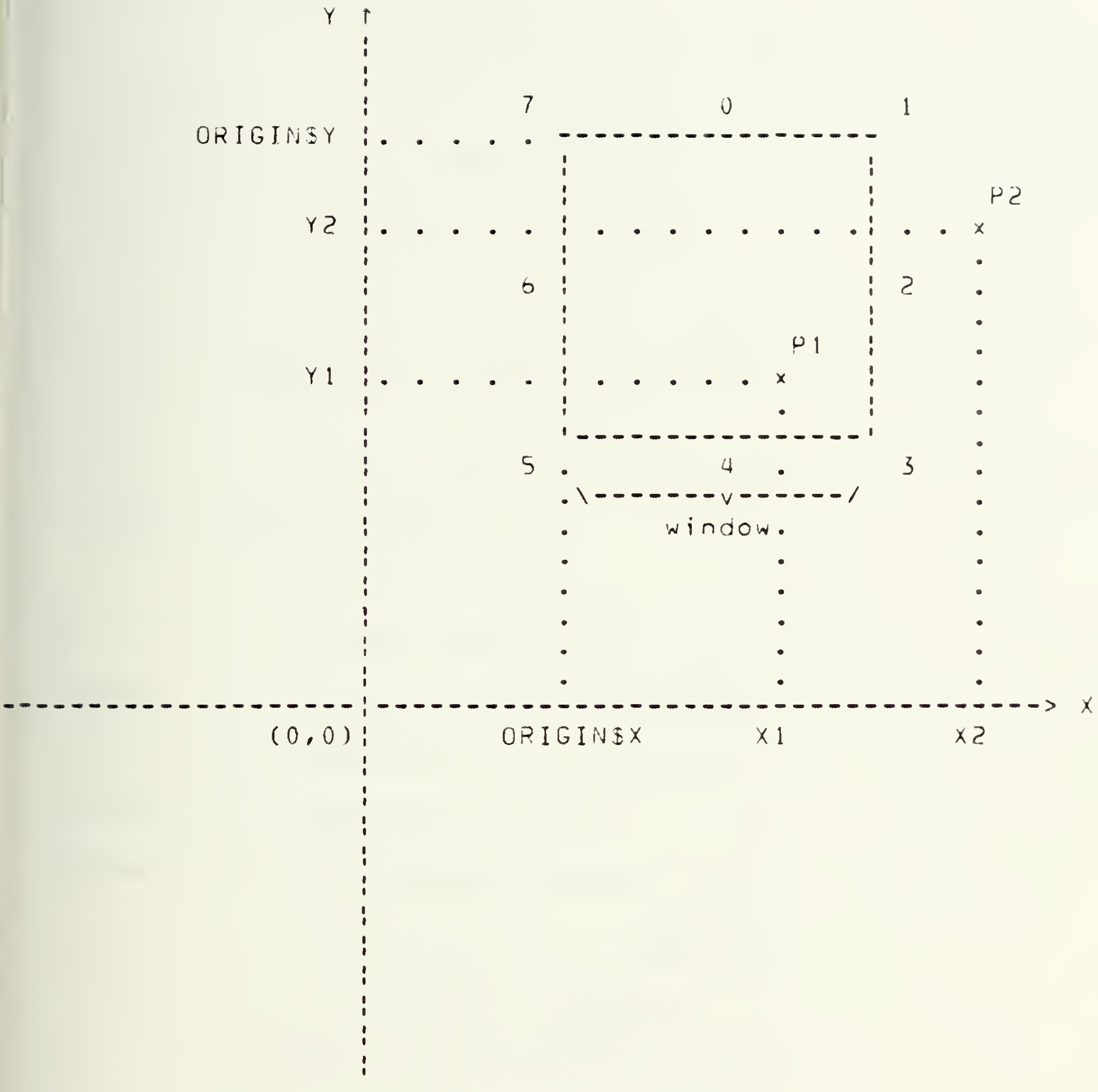


FIGURE NO. 13 WINDOWING SCHEMATIC

possible to plot a given position (X/Y values) in the region covered by the Plasma Panel; thus, the following algorithm was developed.

In Figure No. 13 the point named P1 can be plotted in, but the point P2 can not. A point can be displayed at the Plasma Panel when the coordinates X/Y of that point follow the rules below:

ORIGIN$X + WINDOW > X > ORIGIN$X, and

ORIGIN$Y - WINDOW < Y < ORIGIN$Y

Notice that all those values are in floating-point representation.

c. Normalization

Before displaying a given point in the Plasma Panel it is necessary first to check if the position could be plotted, and secondly to normalize its value to the range specified (addressable matrix - 512 x 512); the first is done as explained above, and the second is as follows:

(1) DELTA$X = X - ORIGIN$X

DELTA$Y = ORIGIN$Y - Y

(2) Take the absolute values of

DELTA$X and DELTA$Y:

DELTA$X = ABS (DELTA$X)

DELTA$Y = ABS (DELTA$Y)

(3) TEMP$X = (511.0 / WINDOW) x DELTA$X

TEMP$Y = (511.0 / WINDOW) x DELTA$Y

(4) Truncate and convert to

integer representation:

X = INTEGER (TEMP$X)

Y = INTEGER (TEMP$Y)

d. Plasma Reorientation

Besides setting a scale for the "window", three ways of positioning the "window" in the coordinate grid system were implemented.

By default, every time the scale has to be changed, the last position of the own ship will be set at the center of the "window"; this is accomplished by setting:

ORIGIN$X = OWN$SHIP$X - HALF$WINDOW$Y, and

ORIGIN$Y = OWN$SHIP$Y - HALF$WINDOW

Notice that the own ship's last position can be set at the center of the "window" any time the operator wants to do so.

The second method implemented was to set the last known position of any contact at the center of the "window"; this was obtained by setting:

ORIGIN$X = CONTACT$POSI$X - HALF$WINDOW, and

ORIGIN$Y = CONTACT$POSI$Y - HALF$WINDOW

In case of no contact being maintained by the system, this method will be ignored by the system, even if requested.

The third method implemented was to set one of 8 fixed positions (refer to Fig. No. 13) at the center of the "window" as requested by the operator; this was obtained by setting:

(1) Point 0:

ORIGIN$X = ORIGIN$X, and

ORIGIN$Y = ORIGIN$Y + HALF$WINDOW

(2) Point 1:

ORIGIN$X = ORIGIN$X + HALF$WINDOW, and

ORIGIN$Y = ORIGIN$Y + HALF$WINDOW

(3) Point 2:

ORIGIN$X = ORIGIN$X + HALF$WINDOW, and

ORIGIN$Y = ORIGIN$Y

(4) Point 3:

ORIGIN$X = ORIGIN$X + HALF$WINDOW, and

ORIGIN$Y = ORIGIN$Y - HALF$WINDOW

(5) Point 4:

ORIGIN$X = ORIGIN$X, and

ORIGIN$Y = ORIGIN$Y - HALF$WINDOW

(6) Point 5:

ORIGIN$X = ORIGIN$X - HALF$WINDOW, and

ORIGIN$Y = ORIGIN$Y - HALF$WINDOW

(7) Point 6:

ORIGIN$X = ORIGIN$X - HALF$WINDOW, and

ORIGIN$Y = ORIGIN$Y

(8) Point 7:

ORIGIN$X = ORIGIN$X - HALF$WINDOW, and

ORIGIN$Y = ORIGIN$Y + HALF$WINDOW

C. TRANSCENDENTAL FUNCTIONS

Three transcendental functions were necessary in solving some problems by the system. These functions were sine and

sine of a given angle, and arc tangent of the ratio of two ven values.

The main goals were the minimum amount of storage for e work area and the minimum execution time in performing e calculations; for these reasons, the Hastings proximations were chosen with slight modifications made to e algorithms suggested in Ref No. 4.

1. Cosine and Sine Functions

As described in the Appendix E, the procedure OS$SIN" performs the cosine and sine of a given angle (in dians); the following steps were taken in the development the algorithm:

a. Save the actual value of the angle

b. Set angle to be between 0 and 2 x PI radians

c. Check for special cases - 90, 270,
and 360 degrees

d. Normalize the angle for the interval 0 and 90
degrees, and save quadrant of the
original angle

e. Convert angle to semicircle units

$$A = ANGLE / PI$$

where PI = 3.141593

f. Perform Hastings approximation

$$Z = (C1 + A^2 (C2 + A^2 (C3 + A^2 (C4 + A^2 (C5 + A^2 \times C6)))))A + A$$

where:

C1 = 0.5707963267949

C2 = -0.6459640964727

C3 = 0.0796926087138

C4 = -0.0046816668674

C5 = 0.0001602588415

C6 = -0.0000034333379

g. Compute cosine and sine:

$$\cos(\text{ANGLE}) = 1.0 - 2.0 \times Z^2$$

$$\sin(\text{ANGLE}) = \sqrt{1.0 - \cos^2(\text{ANGLE})}$$

h. Restore signs for sine and cosine according to the quadrants saved in d.

2. Arc Tangent Function

As described in Appendix E, the procedure "ARC$TAN" performs the arc tangent function of a given ratio (Y/X) of 2 parameter values; the following steps were taken in the development of the algorithm:

a. Save the actual values of the parameters

b. Save sign of parameters to determine quadrant

c. Check for valid arguments (X and Y)

(1) If X = 0 and Y = 0:

Function undefined

(2) If X = 0 and Y = 0:

ANGLE = 90 degrees (for $Y > 0$)

ANGLE = 270 degrees (for $Y < 0$)

d. Form Z to perform the Hastings approximation

$$Z = \frac{|Y| - |X|}{|Y| + |X|}$$

e. Perform the Hastings approximation

$$\text{ANGLE} = (C1 + Z^2 (C2 + Z^2 (C3 + Z^2 (C4 + Z^2 (C5 + Z^2 (C6 + Z^2 (C7 + Z^2 \times C8))))))))Z + PI/4$$

where:

$C1 = 0.9999993329$

$C2 = -0.3332985605$

$C3 = 0.1994653599$

$C4 = -0.1390853351$

$C5 = 0.0964200441$

$C6 = -0.0559098861$

$C7 = 0.0218612288$

$C8 = -0.0040540580$

$PI = 3.141593$

f. Restore angle to proper quadrant

D. POSITIONAL DATA CONVERSION

In the design of the system all the positions can be referred either as latitude and longitude, or as X/Y coordinates; for this reason, some algorithms were developed in order to obtain one or another kind of positional data.

1. Convert LAT and LONG to X/Y Coordinates

The whole system was based in a Coordinate Grid ystem whose origin values were given in terms of latitude nd longitude, and any position in it had an X/Y coordinate defined in relation to the origin; thus, given the values of atitude and longitude of a certain position, it might be converted to that Coordinate Grid System units; i.e., to convert to X/Y coordinates. This was obtained by doing:

a. Compute mean latitude:

```
MEAN$LAT = (SYSTEM$LAT + LAT) / 2.0
```

b. Compute X/Y coordinates:

```
X = (LONG - SYSTEM$LONG) x cos(MEAN$LAT)
Y = LAT - SYSTEM$LAT
```

2. Convert a Given Position in Terms of Bearing and Range from Own Ship to X/Y Coordinates

In order to determine the X/Y coordinates of a position when it is given in terms of bearing and range from the own ship, the following steps were done:

a. Save value of bearing:

```
ANGLE = BEARING
```

b. Compute DELTA$X and DELTA$Y:

```
DELTA$X = RANGE x sin(ANGLE)
DELTA$Y = RANGE x cos(ANGLE)
```

c. Compute X and Y:

```
X = OWN$SHIP$X + DELTA$X

Y = OWN$SHIP$Y + DELTA$Y
```

3. Convert X/Y Coordinates of a Given Position into Latitude and Longitude

a. Compute latitude:

```
LAT = Y + SYSTEM$LAT
```

b. Compute mean latitude:

```
MEAN$LAT = (SYSTEM$LAT + LAT) / 2.0
```

c. Compute longitude:

```
LONG = X / cos(MEAN$LAT) + SYSTEM$LONG
```

APPENDIX B SOFTWARE CATEGORIZATION

MODULES DESCRIPTION

As mentioned before the system was developed around 12 ic modules; there is another module (EXECUTIVE) which is edded in the module MAIN$MODULE.

As shown in Appendix E, every procedure has a comment der which explains what it performs, the parameters with meaning, and, when proper, the usage of that procedure.

MODULES INTERACTION

Due to the interaction capability between modules as owed by the language PL/M 80 through the use of the ributes PUBLIC and EXTERNAL for the procedures, the lowing list was written in order to show this eraction. This list shows the modules which have cedures called by the listed module.

1. MAIN$MODULE:
    a. EXECUTIVE$COMMANDS
    b. DISPLAY$CMDS
    c. COMMANDS
    d. PLASMA$MODULE
    e. CRT
    f. FLTASCII
    g. FLOATING$POINT
    h. TIME
    i. BASICS

2. EXECUTIVE (embedded in MAIN$MODULE):

   a. MAIN$MODULE

   b. EXECUTIVE$COMMANDS

   c. DISPLAY$CMDS

   d. PLASMA$MODULE

   e. CRT

   f. TIME

   g. PLASMA$PRIMITIVES

   h. BASICS

3. EXECUTIVE$COMMANDS:

   a. MAIN$MODULE

   b. EXECUTIVE$COMMANDS

   c. CPA$MODULE

   d. COMMANDS

   e. PLASMA$MODULE

   f. CRT

   g. FLTASCII

   h. FLOATING$POINT

   i. TIME

   j. BASICS

4. CPA$MODULE:

   a. EXECUTIVE$COMMANDS

   b. CPA$MODULE

   c. COMMANDS

   d. FLOATING$POINT

5. DISPLAY$CMDS:

   a. EXECUTIVE$CMDS

    b. CPA$MODULE
    c. DISPLAY$CMDS
    d. COMMANDS
    e. CRT
    f. FLOATING$POINT
    g. BASICS

6. COMMANDS:
    a. COMMANDS
    b. CRT
    c. FLTASCII
    d. FLOATING$POINT
    e. BASICS

7. PLASMA$MODULE:
    a. EXECUTIVE$COMMANDS
    b. COMMANDS
    c. PLASMA$MODULE
    d. CRT
    e. FLOATING$POINT
    f. PLASMA$PRIMITIVES
    g. BASICS

8. CRT:
    a. CRT
    b. BASICS

9. FLTASCII:
    a. FLTASCII
    b. FLOATING$POINT

10. FLOATING$POINT:

    - a. FLOATING$POINT
    - b. BASICS

11. TIME:
    - a. COMMANDS
    - b. CRT
    - c. BASICS

12. PLASMA$PRIMITIVES:
    - a. PLASMA$PRIMITIVES

13. BASICS:
    - a. BASICS

APPENDIX C

OPERATOR'S MANUAL
for the
SURFACE-SUBSURFACE CONTACT PLOTTER SYSTEM
at the
NAVAL POSTGRADUATE SCHOOL

This manual describes the operation of the Surface-Subsurface Contact Plotter System at the Naval Postgraduate School. This manual assumes familiarization with CIC procedures. The specifics about the installation of the equipment required were presented in chapters 4 and 5 and also in Appendix D. The algorithms used were described in Appendix A. All the software required is contained in one diskette labeled PLASMA GEOGRAPHIC PLOTTER PACKAGE: SYSTEM.APL. Figure No. 14 presents a view of how the equipment is typically set up.

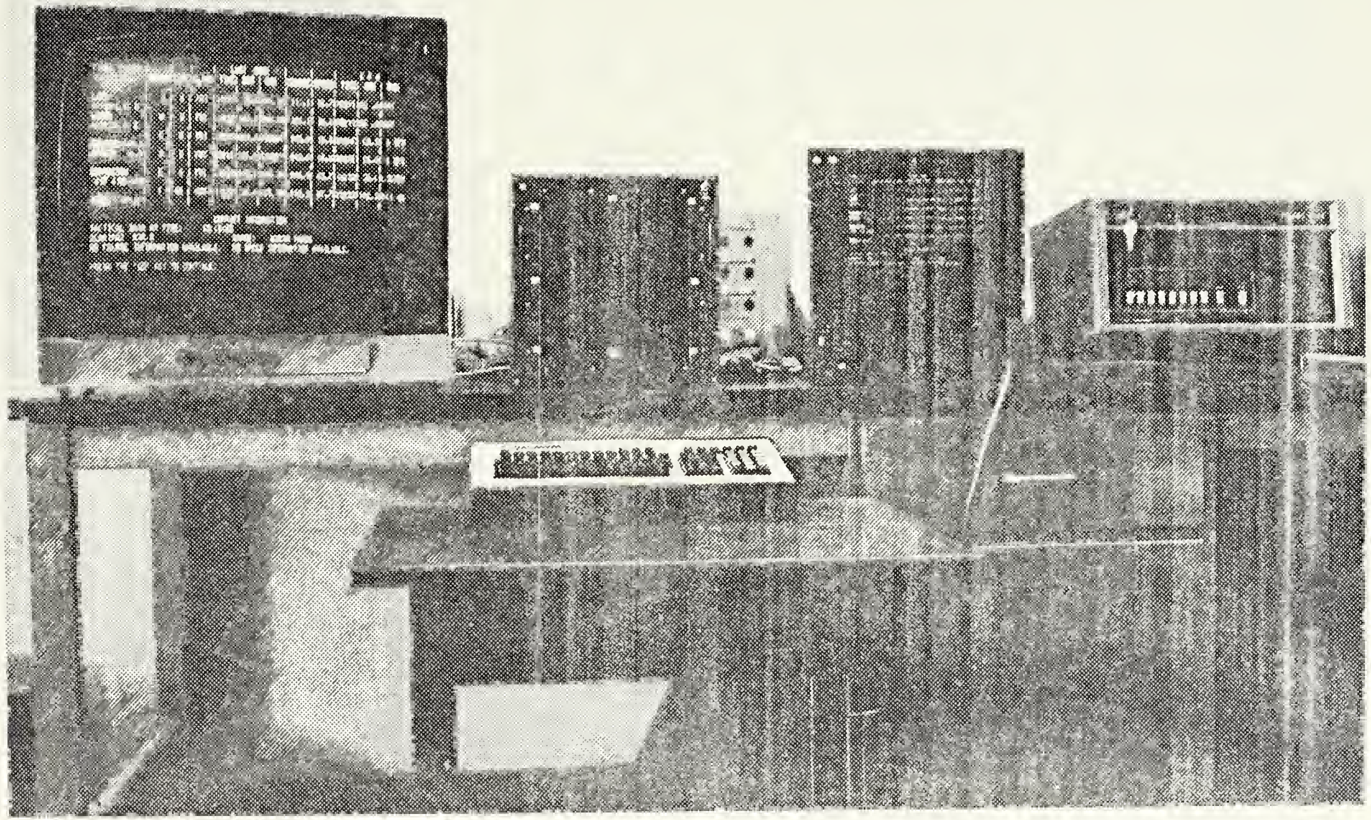


FIGURE NO. 14 EQUIPMENT VIEW

I. STARTUP PROCEDURES

AUTION: NEVER turn on or off the diskette drive with a diskette inserted !!!.

. Turn on MDS system: use the key located at the upper left corner of the front panel and turn it clockwise. The POWER indicator should light.

2. Turn on diskette drive: use POWER switch located at the front panel. The ON indicator should light.

3. Turn on DATAMEDIA Video Terminal (CRT): use switch located on right side. The cursor should appear at the screen after a few seconds. Ensure that the lights CD, CTS, ROLL and FULL DUPLEX are on.

4. Turn on power supply to Plasma Unit. This external power supply should be set at + 5 Volts DC. A red indicator should light.

5. Turn on AN/UYQ-10 Plasma Display Unit: use POWER switch located at front of unit. The indicator located at the upper left corner should light.

6. Turn on any other slave displays, if existing.

7. Place diskette labeled PLASMA GEOGRAPHIC PLOTTER PACKAGE: SYSTEM.APL in drive 0, with the read/write access slot first. Close door of the drive after diskette insertion.

8. Bootstrap the ISIS-II Operating System:

   a. Press top of Intellec BOOT switch.

b. Press top of RESET switch.

c. Observe that INTERRUPT 2 indicator goes on before proceeding.

d. Press space bar of DATAMEDIA Video Terminal keyboard.

e. Observe that INTERRUPT 2 indicator goes off before proceeding.

f. Press bottom of BOOT switch.

g. Observe that the following message appears at the DATAMEDIA Video Terminal screen:

```
ISIS-II,V2.2

-
```

9. Issue the following command:

```
DRTSYS <carriage return>
```

10. After a few seconds, the DATAMEDIA Video Terminal screen should be cleared and then filled with the working format; also, the message "ON LINE." should appear at the AN/UYQ-10 Plasma Display screen.

11. Observe that the slave displays (if existing), are presenting the same information as their respective masters.

12. Follow the instructions for SYSTEM INITIALIZATION as prompted and according to the format explained in the following pages.

13. Notice that the TIME value entered during SYSTEM INITIALIZATION should be that one desired as starting time (the time at which the GO key is depressed, after

the SYSTEM INITIALIZATION mode is completed).

14. During operation, the INTERRUPT 1 indicator should light (after the GO key has been depressed to start the system).

II. SHUTDOWN PROCEDURE

CAUTION: NEVER turn on or off the diskette drive

with a diskette inserted !!!.

1. Press the INTERRUPT 0 switch. The associated indicator should light.
2. Eject the diskette in drive 0.
3. Turn off the equipment in the following order:
    a. Slave displays (if existing).
    b. AN/UYO-10 Plasma Display Unit.
    c. AN/UYO-10 Plasma Display Unit power supply.
    d. DATAMEDIA Video Terminal.
    e. Diskette drive.
    f. Intellec MDS system.

III. FORMATS AND COMMANDS DESCRIPTION

The following pages describe the data elements, input commands, and display commands required for the operation of the SURFACE-SUBSURFACE CONTACT PLOTTER SYSTEM.

NAME:

Time Zone Number: parameter defining the time zone number being used to determine the local time.

FORMAT:

snn where:
s - sign (+ or -).
nn - two digit number.

RANGE:

00 <= nn <= 12

COMMENTS:

Used only for display purposes.

NAME:

Time: parameter defining a time value. Consists of hours, minutes and seconds.

FORMAT:

```
hh:mm:ss   where:
                 hh - value of hours. Two digits.
                 mm - value of minutes. Two digits.
                 ss - value of seconds. Two digits.
```

RANGE:

```
00 <= hh <= 23
00 <= mm <= 59
00 <= ss <= 59
```

COMMENTS:

Once the time is set, the system will maintain the current time and update the time value displayed at the Video Terminal every second.

NAME:

Time Between Updates: parameter that defines the interval of time used by the system to update the geographical position of the own ship.

FORMAT:

sss where:
sss - value in seconds to be used. Three digits.

RANGE:

15 <= sss <= 250

COMMENTS:

Initially, the Time Between Updates is set automatically by the system to 180 seconds. The system will calculate the new position of the own ship and display this value at the Video Terminal and will plot, if possible, the new position at the Plasma Video any time the interval of time between the current time and the actual time is greater than or equal to the Time Between Updates parameter.

NAME:

Course: parameter that determines the general direction at which the own ship or any contact is steering.

FORMAT:

ddd.d where:
ddd.d - value in degrees and tenths of degrees. Three digits.

RANGE:

000.0 <= ddd.d <= 359.9

COMMENTS:

Needs to be entered for the own ship. The system will calculate its value in the case of a contact, although it could also be input for the system if known. Note that the system will eventually override this information, after solving the course problem.

NAME:

Speed: parameter that determines the velocity at which the own ship or any contact is moving.

FORMAT:

kk.k where:
kk.k - value in knots (nautical miles / hour) and tenths of knots. Three digits.

RANGE:

00.0 <= kk.k <= 99.9

COMMENTS:

Needs to be entered for the own ship. The system will calculate its value in the case of a contact, although it could also be input for the system if known. Note that the system will eventually override this information, after solving the speed problem.

NAME:

Bearing: parameter that determines the true bearing of a contact from the own ship, at a given time.

FORMAT:

```
bbb.b    where:
               bbb.b - value in degrees and tenths of
                       degrees. Four digits.
```

RANGE:

000.0 <= bbb.b <= 359.9

COMMENTS:

None.

NAME:

Range: parameter that determines the distance between the own ship and any contact, at a given time. Can be given in yards or in nautical miles.

FORMAT:

mmm.m or yyyyyy where:
mmm.m - value in miles and tenths of miles. Four digits.
yyyyyy - value in yards. Six digits.

RANGE:

000.0 <= mmm.m <= 100.0
000000 <= yyyyyy <= 999999

COMMENTS:

When the system is displaying a Range parameter, all values less than or equal to 5.0 nautical miles will be always displayed in yards. The system keeps the internal value in miles.

NAME:

Latitude: parameter that defines the geographical position of the own ship or any contact. Consists of a sign, degrees, minutes and tenths of minutes.

FORMAT:

dd:mm.m s where:

dd - value in degrees. Two digits.
mm.m - value in minutes and tenths of minutes. Three digits.
s - sign (North (N) or South (S)).

RANGE:

00 <= dd <= 89
00.0 <= mm.m <= 59.9

COMMENTS:

The initial position of the own ship is initially required for the system; later positions of the own ship and all positions for the contacts will be automatically calculated by the system.

NAME:

Longitude: parameter that defines the geographical position of the own ship or any contact. Consists of a sign, degrees, minutes and tenths of minutes.

FORMAT:

```
ddd.mm.m s    where:
                    ddd  - value in degrees. Three digits.
                    mm.m - value in minutes and tenths of
                           minutes. Three digits.
                    s    - sign (East (E) or West (W)).
```

RANGE:

00 <= ddd <= 179
00.0 <= mm.m <= 59.9

COMMENTS:

The initial position of the own ship is initially required for the system; later positions of the own ship and all positions for the contacts will be automatically calculated by the system.

NAME:

X: parameter that defines the position of the own ship or any contact in the Coordinate Grid System being used. Its value is given in miles.

FORMAT:

snnnnnnnnnn.nn where:
s - sign (+ or -).
nnnnnnnnnn.nn - value in miles and hundreths of miles. Twelve digits.

RANGE:

0.0 <= |nnnnnnnnnn.nn| <= 9999999999.99

COMMENTS:

Is not determined directly by the operator; it is calculated automatically by the system which also takes care of its format and range. Could be positive or negative.

AME:

Y: parameter that defines the position of the own ship r any contact in the Coordinate Grid System being used. Its alue is given in miles.

ORMAT:

nnnnnnnnnn.nn where:
s - sign (+ or -).
nnnnnnnnnn.nn - value in miles and hundreths of miles. Twelve digits.

ANGE:

0.0 <= |nnnnnnnnnn.nn| <= 9999999999.99

OMMENTS:

Is not determined directly by the operator; it is calculated automatically by the system which also takes care of its format and range. Could be positive or negative.

NAME:

Designation (Desig): parameter that defines the name given to a particular contact. Consists of two alphabetic characters or one blank and one alphabetic character.

FORMAT:

Aa where:
A - Any alphabetic character or a blank.
a - Any alphabetic character.

RANGE:

Not applicable.

COMMENTS:

Needs to be unique for the set of contacts maintained by the system at any given time. Upper or lower case characters can be used, although the system will change all lower case characters to their upper case equivalents.

NAME:

Type: parameter that defines if the contact is of surface or sub-surface kind.

FORMAT:

t where:
t - could be S (Surface) or SS (Sub-surface).

RANGE:

Not applicable.

COMMENTS:

None.

NAME:

Class: parameter that defines the identity and purpose of any contact.

FORMAT:

c where:
c - can have three values:
F if Friendly (FRI).
H if Hostile (HOS).
U if Unknown (UNK).

RANGE:

Not applicable.

COMMENTS:

Depending on the Class of any contact, the symbol used to plot its positions at the Plasma Display will vary: a circle is used for Friendly contacts, and an 'X' is used for Hostile and Unknown contacts.

AME:

Scale: parameter that defines the scale at which the iture presented at the Plasma Unit is displayed. Can be ecified up to one hundredth of a mile/inch.

RMAT:

mm where:
mm.mm - value in miles and hundreths of miles per inch. Four digits.

NGE:

00.25 <= mm.mm <= 25.00

MMENTS:

The Scale value is used in determining the size of the ndow used in forming the picture to be displayed on the asma Unit.

AME:

Safe CPA Range: parameter that defines the radius of a ircle with center at the own ship; any contact that will ass through this security circle will be considered in ollision.

ORMAT:

yyy where:
yyyy - value in yards. Four digits.

ANGE:

0050 <= yyyy <= 1000

OMMENTS:

Initially the Safe CPA Range parameter is set utomatically by the system to a value of 0050 yards.

NAME:

Wind Direction: this parameter indicates the true bearing from which the wind is blowing.

FORMAT:

ddd.d where:
ddd.d - value in degrees and tenths of degrees. Four digits.

RANGE:

000.0 <= ddd.d <= 359.9

COMMENTS:

Used only for display purposes.

AME:

Wind Speed: this parameter indicates the speed at which he wind is blowing.

ORMAT:

kk.k where:
kk.k - value in knots (nautical miles / hour) and tenths of knots. Three digits.

RANGE:

00.0 <= kk.k <= 99.9

COMMENTS:

Used only for display purposes.

AME:

Origin Update.

ESCRIPTION:

This command is used to modify the Coordinate Grid Origin arameters.

NPUT REQUIRED:

- New Latitude and Longitude values.

PTIONAL INPUT:

- None.

OMMENTS:

This command causes the system to change all the X/Y alues that had been calculated, and also to redraw the icture represented at the Plasma Display with the last osition of the own ship at the center.

NAME:

Own Ship Update.

DESCRIPTION:

This command is used to modify the parameters of the own ship: Latitude, Longitude, Course and Speed, in a selective way.

INPUT REQUIRED:

- When prompted by the system, indicate which parameters are going to be changed.

OPTIONAL INPUT:

- New Latitude value.
- New Longitude value.
- New Course value.
- New Speed value.

DESCRIPTION:

When this command is issued and when the system prompts the user to indicate which parameters will be changed, if the user responds that no parameter is desired to be changed, then the control will be passed to the system again.

The time information needed will be automatically obtained by the system, and will be the time at which this command was issued.

If any parameter is changed, this causes the system to automatically update the position of the own ship to the moment at which the command was issued.

All changes made will be displayed at the Video Terminal and reflected in the Plasma Display.

If the Latitude and/or Longitude values are changed, then all X/Y values in the system will be recalculated and a new picture with the new position of the own ship at the center will be presented at the Plasma Display.

Notice also that when the Course and/or Speed parameters are changed, no CPA information will be available for any contact until a new mark is obtained for any contact from

which CPA information is desired.

NAME:

Create.

DESCRIPTION:

This command is issued to record a new contact.

INPUT REQUIRED:

- Designation.
- Type.
- Class.
- Bearing.
- Range.

OPTIONAL INPUT:

- Course and Speed values if known.

COMMENTS:

When this command is issued, the position of the own ship is updated to that moment; the first position of the new contact will be displayed at the Plasma Unit, if possible (if it is inside the window).

The system will check for the Designation given to the new contact, to make sure that it is not being utilized at that moment; if such an error is detected, a warning message will be displayed.

The time information needed for the contact will be automatically obtained by the system, and will be the time at which this command was issued.

Notice that when the system already has 15 contacts, this command will not be accepted, and a warning message will be issued.

If the system is maintaining less than six contacts, the information entered for the new contact will be automatically displayed at the Surface Status Board presented in the Video Terminal. In the case of six or more existing contacts, no information will be automatically displayed.

NAME:

Remove Contact.

DESCRIPTION:

This command is used to remove a contact from the system.

INPUT REQUIRED:

- Assurance that this command is really desired.

OPTIONAL INPUT:

- Designation of the contact desired to be removed.
- Designation of a contact desired to be displayed.

COMMENTS:

The system will prompt the user to determine if this command is really needed; if the user made a mistake in calling this command this is the moment to rectify it. If the user agrees that this command is really needed, then the system will prompt for the designation of the contact desired to be deleted.

If the designation given to the system does not exist, the system will issue a warning message, and will again try to determine if this command is really desired.

If the contact removed was being displayed at the Surface Status Board, its information will be erased; if at this moment all the contacts in the system are being displayed at the Surface Status Board, then the control will pass again to the system; however, if not all the contacts were being displayed, the system will prompt for the Designation of a contact that the user desires to be displayed in place of the recently removed contact; the system at this point will check that the Designation given really exists and that it does not correspond to a contact already being displayed; if any error is discovered, a warning message will be issued.

If the contact just removed was not being displayed at the Surface Status Board, the control will pass automatically to the system.

If the contact just removed was being displayed at the Plasma Unit, its position(s) will not be erased until a new

picture is drawn.

The information about the class of the contacts in the system will be also automatically updated.

This command will not be accepted if there are no contacts in the system, and a warning message will be issued.

NAME:

Redesignate.

DESCRIPTION:

This command is used to give a new Designation to any contact already in the system.

INPUT REQUIRED:

- Old Designation.
- New Designation.

OPTIONAL INPUT:

- None.

COMMENTS:

This command will check for both Designations to be correct. If any one of them represents an error - the Old Designation does not exist or the New Designation already exists - a warning message will be issued. The Old Designation will not be accepted as New Designation at the same time.

If the contact being redesignated is displayed at the Surface Status Board, its DESIG field will be automatically updated.

If the contact being redesignated is displayed at the Plasma Unit, its DESIG field will not be automatically updated until the next time the picture is drawn.

This command will not be accepted if there are no contacts in the system, and a warning message will be issued.

NAME:

Contact Update.

DESCRIPTION:

This command is used to update the information about any contact being maintained by the system: Type, Class, Bearing, Range, Course and Speed, may be changed selectively.

INPUT REQUIRED:

- Designation of the contact desired to be updated.
- Indication of which parameters are desired to be updated.

OPTIONAL INPUT:

- New Type value.
- New Class value.
- New Bearing value.
- New Range value.
- New Course value.
- New Speed value.

COMMENTS:

If the user responds that no parameter is desired to be changed, the control will pass automatically to the system.

The system will prompt for only those parameters that were indicated by the user.

The time information needed for the contact will be automatically obtained by the system, and will be the time at which this command was issued.

If the Bearing and/or Range parameters are changed, this will cause the system to automatically update the position of the own ship to the moment at which this command was issued.

The system will check that the Designation given is correct, and if an error is detected, then a warning message will be issued.

If the contact being updated is also displayed at the Surface Status Board, its new parameters will be displayed

after the command finishes.

The new position determined will be displayed at the Pasma Unit if possible.

This command will not be accepted if there are no cntacts in the system, and a warning message will be isued.

NAME:

Swap Contacts.

DESCRIPTION:

This command is used to change the list of contacts that are being displayed on the Surface Status Board.

INPUT REQUIRED:

- Designation of a contact desired to be out of the display.
- Designation of a contact desired to be in the display.

OPTIONAL INPUT:

- None.

COMMENTS:

The system will check for the following possible errors:

- A non-existent Designation is given.
- The Designation of a contact that is not at the display is given to indicate the contact desired to be out of the display.
- The Designation of a contact already in the display is given to indicate the new contact desired to be in the display.

In any of these cases, the system will issue a warning message and continue to prompt.

This command will not be accepted if the number of contacts in the system is less than seven, and a warning message will be issued.

ME:

Time.

SCRIPTION:

This command is used to update/change all the parameters at the system has with respect to time: Time Zone Number, stem Clock Value and Time Between Updates.

PUT REQUIRED:

- Indication of which parameters are desired to be dated.

TIONAL INPUT:

- New Time Zone Number value.
- New System Clock value.
- New Time Between Updates.

MMENTS:

If the user responds that no parameter is desired to be anged, the control will pass automatically to the system.
If the Time Zone Number is modified, the new value will automatically displayed at the Video Terminal.
If the Time Between Updates is modified and the new value smaller than the old one, the elapsed time from the last tomatic update of the own ship's position will be compared ainst the new Time Between Updates, and if appropriate, e position of the own ship will be updated.

NAME:

CPA Safe Range Update.

DESCRIPTION:

This command is used to update/change the value of the PA Safe Range parameter.

Remember that the initial default value of the CPA Safe ange is 0050 yards.

NPUT REQUIRED:

- New CPA Safe Range value.

PTIONAL INPUT:

- None.

OMMENTS:

The new CPA Safe Range will be used in all following CPA alculations, and the Display will not be affected to eflect this change until such new calculations occur.

command input command

nd Update.

IPTION:

is command is used to introduce/update information
the wind.

REQUIRED:

New Wind direction value.
New Wind speed value.

NAL INPUT:

None.

NTS:

is information is used for display purposes only.

NAME:

Scale Update.

DESCRIPTION:

This command is used to modify the value of the Scale parameter being used to define the window that limits the picture to be represented at the Plasma Display.

INPUT REQUIRED:

- New Scale value.

OPTIONAL INPUT:

- None.

COMMENTS:

This command will cause the system to define a new window to be used in forming the picture to be represented at the Plasma Display; a new picture will be presented reflecting the change made, with the last position of the own ship at the center.

Notice that the value of the Scale parameter is permanently displayed at the Plasma Unit.

NAME:

Plasma Reorient.

DESCRIPTION:

This command is used to redefine the position of the window used to form the picture to be presented at the Plasma Display.

INPUT REQUIRED:

- Type of reorientation desired.

OPTIONAL INPUT:

- Value of new point to be at the center of the new window.
- Designation of a contact whose last position is desired to be at the center of the new window.

COMMENTS:

This command allows the user to redefine the position of the window in three different ways:

- By selecting one of eight predefined points to be the center of the new window.
- By making the last position of the own ship to be the center of the new window.
- By selecting the last position of any contact to be the center of the new window. This possibility is allowed only when there is at least one contact in the system. The system will also check for a valid Designation, and will issue a warning message if necessary.

This command will cause the system to draw a new picture at the Plasma Display, according to the picture defined by the new position of the window.

NAME:

Origin.

DESCRIPTION:

This command is used to display the values of the Coordinate Grid Origin: Latitude and Longitude.

INPUT REQUIRED:

- None.

INFORMATION DISPLAYED:

- Coordinate Grid Origin Latitude value.
- Coordinate Grid Origin Longitude value.

COMMENTS:

This information is displayed using the lower portion of the screen.

The control will pass to the system after pressing the 'GO' key when indicated.

ME:

Scale.

SCRIPTION:

This command is used to display the value of the Scale rameter currently in use.

PUT REQUIRED:

- None.

FORMATION DISPLAYED:

- Scale parameter value.

MMENTS:

This information is displayed using the lower portion of e screen.
The control will pass to the system after pressing the 0' key when indicated.
Notice that the value of the Scale parameter is rmanently displayed at the Plasma Unit.

NAME:

Own Ship.

DESCRIPTION:

This command is used to display the parameters associated with the last position of the own ship.

INPUT REQUIRED:

- None.

INFORMATION DISPLAYED:

- First page: Positional data.
  1) Latitude and Longitude values.
  2) X and Y values of the last determined position.
- Second page: Tactical data.
  1) Time of last Course and/or Speed change.
  2) Course and Speed in effect at the moment.

COMMENTS:

This information is displayed using the lower portion of the screen.

The different pages are obtained by pressing the 'GO' key each time a new page is desired.

The control will return to the system after the last page has been passed, by pressing the 'GO' key.

ME:

Contact Information.

SCRIPTION:

This command is used to display information about any ntact being maintained by the system.

PUT REQUIRED:

- Designation of a contact whose information is desired.

FORMATION DISPLAYED:

- First page: General data.
    1) Designation, Type, Class, and Current Number of Positions in the system.
- Second page: Positional data.
    1) Latitude and Longitude values of the last determined position.
    2) X and Y values of the last determined position.
- Third page: Tactical data.
    1) Time at which the present data was obtained.
    2) Bearing and Range values of last mark.
    3) Course and Speed values if available.
- Fourth page: CPA data.
    1) Time at which CPA will occur, if applicable.
    2) Bearing and Range defining the CPA, if applicable.
    3) Note: if CPA can not be calculated a message ll be displayed in place of this information. Also, if a ecial CPA case is present, one of the following messages ll be displayed:
        - "COLLISION AT ..(time).." (Blinking).
        - "MOVING AWAY".
        - "SAME COURSE AND SPEED".
- Fifth page: Actual Estimated Position.
    1) Bearing and Range defining the estimated sition of the contact. This will only be displayed if the urse and Speed values of the contact are known.

MMENTS:

This information is displayed using the lower portion of the screen.

The different pages are obtained by pressing the 'GO' key each time a new page is desired.

The control will return to the system after the last page has been passed, by pressing the 'GO' key.

The system will check for a valid Designation, and a warning message will be issued if a mistake is detected.

This command will not be accepted if there are no contacts in the system, and a warning message will be issued.

AME:

Contacts in System.

DESCRIPTION:

This command is used to obtain information about the Designation of all the contacts in the system.

INPUT REQUIRED:

- None.

INFORMATION DISPLAYED:

- Designations of the contacts maintained by the system.

COMMENTS:

This information is displayed by using the lower portion of the screen.

This command will not be accepted if there are no contacts in the system, and a warning message will be issued.

NAME:

Request CPA Safe Range.

DESCRIPTION:

This command is used to obtain information about the current value of the CPA Safe Range.

INPUT REQUIRED:

- None.

INFORMATION DISPLAYED:

- Actual CPA Safe Range value.

COMMENTS:

This information is displayed using the lower portion of the screen.

The control will pass to the system after pressing the 'GO' key when indicated.

AME:

Wind.

ESCRIPTION:

This command is used to obtain information about the ind.

NPUT REQUIRED:

- None.

NFORMATION DISPLAYED:

- Wind direction and wind speed values if available. If here is no information about the wind, a warning message ill be issued.

OMMENTS:

This information is displayed using the lower portion of he screen.
The control will pass to the system after pressing the GO' key when indicated.

NAME:

Time Between Updates.

DESCRIPTION:

This command is used to obtain information about the current value of the Time Between Updates parameter.

INPUT REQUIRED:

- None.

INFORMATION DISPLAYED:

- Actual value of the Time Between Updates parameter.

COMMENTS:

This information is displayed using the lower portion of the screen.

The control will pass to the system after pressing the 'GO' key when indicated.

Notice that the default value of this parameter is 180 seconds.

APPENDIX D FLOATING-POINT HARDWARE BOARD

A. GENERAL INFORMATION

The floating-point package developed for this system is based on the SBC 310 High-speed Mathematics Unit from INTEL Corporation. As described by the Reference No. 10, the SBC 310 unit is a member of a complete line of the INTEL SBC 80 system expansion modules. In performing high-speed mathematic functions, the Math Unit acts as an intelligent processor slaved to one or more SBC 80 computer masters. The Math Unit performs its repertoire of 14 arithmetic functions an order of magnitude faster than is possible with software routines.

B. DESCRIPTION OF THE MATH UNIT

The Math Unit is a microprogrammed processor on a single board and is designed to be plugged into a standard SBC 604/614 Modular Backplane and Cardcage to interface directly with an SBC 80 Single Board Computer or to be used with an INTEL Intellec Microcomputer Development System (MDS).

Standard operations include floating-point add, subtract, multiply, divide, square, and square root; fixed-point integer multiply, divide, and extended divide; conversions between fixed and floating-point representations; and test, compare, and argument exchange operations.

The Math Unit implements unbiased rounding for maximum

accuracy. Unbiased rounding is the same as ordinary rounding unless the result is exactly midway between two floating-point numbers; in this case, ordinary rounding always increases the result, whereas unbiased rounding rounds the result to the nearest even number. When a calculation is performed that results in either an exponent underflow or overflow, the Math Unit provides exponent wraparound to prevent loss of information.

Operation codes for invoking the arithmetic functions are passed to the Math Unit via I/O Write commands, which are also used to initialize the unit with a memory base address. I/O Read commands are used to determine the Math Unit Status. Arguments are passed to the Math Unit via Memory Write commands and the results are obtained via Memory Read commands.

The Math Unit, which can be operated either in the Interrupt or Polled mode, generates a Busy signal during processing operations and generates either a Complete signal or an Error signal after the computation is complete. The information to the host computer which these three signals convey is explained in Reference No. 10.

The memory base address and I/O base address are user selectable. The 16-bit memory address is completely under software control and is assigned by the host processor through a sequence of I/O Write commands addressed to the Math Unit. The 8-bit I/O base address is selected by a dual inline package (DIP) switch on the board.

All Math Unit operations, including arithmetic calculations, data flow between functional elements on the board bus interface, and associated logical tasks, are resident microprogram permanently stored in a set of eight INTEL 3604 Eraseable Programmable Read Only Memory (EPROM) chips. This memory provides 1,024 micro-instructions of 32 bits each.

C. PREPARATION FOR USE

1. Installation Considerations

The Math Unit board is designed for interface with an INTEL SBC 80 Single Board Computer based system or an INTEL Intellec Microcomputer Development System (MDS).

When installing the SBC 310 in an INTEL Intellec MDS, the CPU board needs to be reconfigured in order to generate a Qualified Write Signal; this reconfiguration was obtained by overriding the advanced acknowledge (AACK) feature, by moving a jumper labeled "advanced write" from a D-C connection to an E-D connection and by disabling the AACK/line (pin 25) on the CPU board as shown in the schematic diagram at page 3-47 of the Reference No. 6.

Other details about installation can be seen on Chapter 2 of the Reference No. 10.

2. I/O Base Address Switches

The host processor transmits control information and receives status information from the Math Unit by issuing I/O Write and I/O Read commands, respectively. The I/O address used for these commands is relative to an 8-bit base

address that must be a multiple of 8. This base address is assigned by the user by means of an 8-pole dual inline package (DIP) switch assembly. Five of the eight switch poles are connected to the I/O base address detection logic; the other three poles are unused.

The Math Unit used had its DIP switch set to the I/O base address of 10 hexadecimal (only switch pole no. 4 was set on).

3. Programming Information

The I/O base address, which must be assigned by switch selection before the memory base address can be assigned, is normally performed as part of the initial installation procedure. This switch setting allows the user to establish a reference or base to the ports being used in the I/O operations; Table No. IV shows the configuration of the I/O addressing as it was set in the system.

The memory base address, which is software controlled, is assigned by a sequence of two I/O write commands. The first command is addressed to port P+1 and loads the low-order byte of the memory base address. The second command is addressed to port P+2 and loads the high-order byte of the memory base address. The memory base address must be a multiple of 16; i.e., the lower byte must be in the form X0H (X is any hexadecimal digit) to accomodate the 16 required memory locations used in the arithmetic operations. In the system developed for this thesis the memory base address was set at 0F790H. After both

I/O PORT ADDRESS	OUTPUT	INPUT
P = 010H	OP CODE	R
011H	MEM LOW	STATUS BYTE
012H	MEM HIGH	R
013H	R	R
014H	R	R
015H	R	R
016H	R	R
017H	R	FLAG BYTE

P: I/O base address.
R: Reserved.
OP CODE: Mathematic function; see Table I.
MEM LOW: Memory base address (lower byte).
MEM HIGH: Memory base address (upper byte).

TABLE IV. I/O ADDRESSING

bytes are output, the memory base address (M) is established and need not be reloaded during any subsequent operations. An initialization routine for establishing the memory base address was designed and it can be seen in the "INIT$FP" procedure in the floating-point module (see "FLOATING$POINT" module at Appendix E).

4. Math Unit Functions

The Math Unit performs floating-point arithmetic, fixed-point integer arithmetic, compare and test operations, and float-to-fix and fix-to-float conversions. Operation codes and execution times for the various functions are listed in Table No. I. Arithmetic and conversion formats are shown in Table No. V.

Beyond the functions performed by the Math Unit, the Floating-point package was designed with two more procedures which compute the cosine and sine of a given angle and the arc tangent value of the ratio of two given arguments.

It was observed that the compare operation between two floating-point numbers did not work properly when the two numbers were both negative; due to this fact, further code was implemented in the "FCMPR" procedure as can be seen in the Appendix E.

5. Argument and Result Data Formats

Argument and result data formats and memory locations for the various operations are presented in Table No. VI. For each argument and result, this table includes a FORMAT number cross-referenced to one of the four formats

```
----------------------------------------------------------------
:FORMAT:          SINGLE PRECISION FLOATING POINT               :
:  NO. :                                                        :
:======:========================================================:
:      :                                                        :
:      :        !!!!!!!!!!!!!!!!!!       !!!!!!!!!!!!!!!!!!      :
:      :        s7      1022    16       15      8 7      0      :
:      :         E      EEF      F       F       F F      F      :
:      :        '---v---''---v---'       '---v---''---v---'      :
:   1  :           M+3      M+2             M+1        M         :
:      :                                                        :
:      : where:      M = memory base address.                   :
:      :             S = "0" = positive; "1" = negative.        :
:      :         E7-E0 = biased exponent; bias = 07FH.          :
:      :        F22-F0 = fraction; F is always normalized.      :
:------:--------------------------------------------------------:
:      :               FIXED POINT INTEGER                      :
:      :--------------------------------------------------------:
:      :                                                        :
:      :        !!!!!!!!!!!!!!!!!!                              :
:      :        15       87       0                             :
:      :        F        FF       F                             :
:   2  :        '---v---''---v---'                              :
:      :           M+1      M                                   :
:      :                                                        :
:      : where:      M = memory base address.                   :
:      :        F15-F0 = 16-bit integer (unsigned).             :
:------:--------------------------------------------------------:
:      :                                                        :
:      :        !!!!!!!!!!!!!!!!!!       !!!!!!!!!!!!!!!!!!      :
:      :        31     24 23    16       15      8 7      0      :
:      :        F       F F      F       F       F F      F      :
:  2A  :        '---v---''---v---'       '---v---''---v---'      :
:      :           M+3      M+2             M+1        M         :
:      :                                                        :
:      : where:      M = memory base address.                   :
:      :        F31-F0 = 32-bit integer (unsigned).             :
:------:--------------------------------------------------------:
:      :              CONVERSION FUNCTIONS                      :
:      :--------------------------------------------------------:
:      :                                                        :
:      :        !!!!!!!!!!!!!!!!!!       !!!!!!!!!!!!!!!!!!      :
:      :        s30    24 23    16       15      8 7      0      :
:      :         F      F F      F       F       F F      F      :
:   3  :        '---v---''---v---'       '---v---''---v---'      :
:      :           M+3      M+2             M+1        M         :
:      :                                                        :
:      : where:      M = memory base address.                   :
:      :             S = "0" = positive; "1" = negative.        :
:      :        F30-F0 = two's complement integer.              :
'------'--------------------------------------------------------'
```

TABLE V. ARITHMETIC AND CONVERSION FORMATS

on in Table No. V. Table No. VI also includes the OP CODE each operation. It is important to note that the result n operation replaces the first argument in memory, and a the second argument may be destroyed in the course of e computation; these side effects were avoided in the oting-point software design by saving the original values d by allowing the user the possibility of having one of e operands as the result. Error conditions for each ation are described in the next paragraph.

6. Status and Flags

The Math Unit may be operated in the interrupt mode olled mode.

In the interrupt mode, the Math Unit may be wire- pped to initiate an interrupt request under one or both the following conditions:

a. Operation complete without an error.

b. Operation complete with an error.

These "completion" signals may be individually e-wrapped to separate interrupt lines or both mpletion" signals may be wire-wrapped to the same errupt line [Ref. No. 10].

In the polled mode, the subroutine designed for this pose checks both the status byte and the flag byte bles No. VII and No. VIII). The polled mode procedure ps on testing the busy bit until the busy bit is clear, then checks the error bits. If an error exists, the or code is input from the Math Unit and a message error

OP	OP CODE	ARGUMENT FORMAT*	ARGUMENTS**	RESULT FORMAT*	RESULT***
UL	0	2	M,M+1 M+4,M+5	2A	M,M+1,M+2,M+3
IV	1	1	M,M+1 M+4,M+5	2	M, M+1 (rem. in M+4,M+5)
DIV	E	2A 2	M,M+1,M+2,M+3 M+4,M+5	2A	M,M+1,M+2,M+3 (rem. in M+4, M+5,M+6,M+7)
MUL DIV ADD SUB	2 3 4 5	1	M,M+1,M+2,M+3 M+4,M+5,M+6,M+7	1	M,M+1,M+2,M+3
SQR QRT	6 7	1	M,M+1,M+2,M+3	1	M,M+1,M+2,M+3
TDS	8	3	M,M+1,M+2,M+3	1	M,M+1,M+2,M+3
XSD	9	1	M,M+1,M+2,M+3	3	M,M+1,M+2,M+3
MPR	A	1	M,M+1,M+2,M+3	-	STATUS byte
TST	B	1	M,M+1,M+2,M+3	-	STATUS byte
XCH	F	Any	M,M+1,M+2,M+3 M+4,M+5,M+6,M+7	-	Rotates both arguments.

* Refer to appropriate FORMAT NO column in Table V.

** Second argument is always the operator, and may be destroyed during the operation.

*** Results of all operations, except FIXSD are rounded. FIXSD truncates the result.

TABLE VI. OPERATION ARGUMENT AND RESULT DATA FORMATS

7	6	5	4	3	2	1	0
R	R	R	R	R	E	C	B*

where: R is reserved for future use.
B is busy.
C is operation complete without error.
E is operation complete with error.

* When B = 1, the Math Unit is busy and cannot respond to further requests except requests for flags.

TABLE VII. FLAG BYTE FORMAT

7	6	5	4	3	2	1	0
=	>	<	R	R	ERR		

where: R is reserved for future use.
= is equal (for FCMPR and FZTST).
> is greater than (for FCMPR and FZTST).
< is less than (for FCMPR and FZTST).

ERR is a 3-bit error code specifying one of the following error conditions:

000 No error.
001 Divide by zero.
010 Domain error.
011 Overflow.
100 Underflow.
101 First argument invalid.
110 Second argument invalid.
111 Reserved.

TABLE VIII. STATUS BYTE FORMAT

is issued. The software developed for control of the Math Unit in the system employs the polled mode described.

As mentioned before, the condition of the Math Unit is continuously updated and stored. The flag byte shown in Table No. VII may be obtained by performing an I/O Read command to P+7 (Table No. IV). After an operation is completed, the status byte may be obtained by performing an I/O Read command to P+1 (Table IV).

As shown in Table No. VIII, the status byte indicates error conditions where applicable and the results of Compare ("FCMPR" procedure) and Test ("FZTST" procedure) operations. Each of the six error conditions are defined as follows:

a. Divide by Zero (001) - This error condition is returned by either "DIV", "EDIV", or "FDIV" procedures to indicate that an attempt was made to divide by zero.

b. Domain Error (010) - This error condition is returned by the "FSQRT" procedure to indicate that the argument was not in the domain of the function; i.e., an attempt was made to take the square root of a negative number.

c. Overflow (011) - This error condition is returned by the "FADD", "FSUB", "FMUL", "FDIV", "FSQR", and "FIXSD" procedures. In the case of "FIXSD" procedure, this error indicates that the floating-point number is too large to be converted to a 32-bit two's complement signed integer. If an overflow error occurs during "FIXSD", the floating-

point argument is left unchanged and may be read from the Math Unit.

In all other cases, this error condition signifies that the exponent of the result is too large to be represented in eight bits. In this case, 0BEH is subtracted from the resulting exponent (bringing it back into range for other computations and ensuring a valid result), and the lower eight bits of the exponent are returned in the exponent field of the result.

d. Underflow (100) - This error condition is returned by the "FADD", "FSUB", "FMUL", "FDIV", and "FSQR" procedures to indicate that the exponent of the result is too small to be represented in eight bits. In this case, 0BEH is added to the resulting exponent (bringing it back into range for other computations and ensuring a valid result), and the lower eight bits of the exponent are returned in the exponent field of the result.

e. First Argument Invalid (101) - This error condition is returned by the "FADD", "FSUB", "FMUL", "FDIV", "FSQR", "FSQRT", "FIXSD", "FCMPR", and "FZTSI" procedures to indicate that the first (or only) argument for the specified function is invalid. The second argument (if applicable) is not checked if this error is encountered. The invalid argument is left unchanged and may be read from the Math Unit.

f. Second Argument Invalid (110) - This error condition is returned by the "FADD", "FSUB", "FMUL", "FDIV",

and "FCMPR" procedures to indicate that the second argument for the specified function is invalid. This error condition occurs only after the first argument is checked and found valid. The invalid argument is left unchanged and may be read from the Math Unit.

NOTES:

(1) The floating-point argument may be expressed as:

$(-1) \times (2^{E} - \text{bias}) \times (1.F)$

Notice that a "1" is assumed in the highest position of F.

(2) There is one unique representation for zero:

$S = 0$

$E7\text{-}E0 = 0$

$F22\text{-}F0 = 0$

(3) The following representations are invalid:

(a) assures unique representation of zero

$E7\text{-}E0 = 0$, and

$S \neq 0$ or $F22\text{-}F0 \neq 0$

(b) reserved for future enhancements

E7-E0 = 0FFH

7. Examples of Floating-point Number Representation

As shown in Table No. V the representation of a floating-point number has one particularity that needs to be mentioned; i.e., a "1" is always assumed in the highest bit position, and then it yields an effective 24-bit mantissa. For the sake of clarity some examples are given in Table No. IX.

F.P. NUMBER	M	M+1	M+2	M+3
-5.0	00H	00H	0A0H	0C0H
-3.0	00H	00H	40H	0C0H
-2.0	00H	00H	00H	0C0H
-1.0	00H	00H	80H	0BFH
0.0	00H	00H	00H	00H
+1.0	00H	00H	80H	3FH
+2.0	00H	00H	00H	40H
+3.0	00H	00H	40H	40H
+5.0	00H	00H	0A0H	40H
+5.4	0CDH	0CCH	0ACH	40H

TABLE IX. FLOATING-POINT NUMBERS

APPENDIX E PROGRAM LISTINGS

. EXTERNAL DECLARATIONS

1. EXTER.............................. 157

2. EXTER$1.............................. 167

3. EXTER$2.............................. 172

. PROCEDURES BY MODULE

1. MAIN$MODULE.............................. 174

NO$WIND.............................. 179

NO$CONTACT.............................. 180

NOT$ENOUGH$CONTACTS.............................. 181

TOO$MANY$CONTACTS.............................. 182

MOVE$OWN$SHIP.............................. 183

EXECUTIVE (not a procedure).............................. 186

2. EXECUTIVE$COMMANDS.............................. 193

DE$HASH.............................. 200

CHECK$GO$KEY.............................. 201

DISPLAY$KIND.............................. 202

CHECK$DESIG.............................. 203

CONV$MIN$RAD.............................. 204

CONV$RAD$MIN.............................. 205

CONV$XY.............................. 206

CONV$REL$XY.............................. 208

INIT$STRUCTURES.............................. 210

GET$SYSTEM$PARAMETERS.............................. 212

DISPLAY$CONTACT.............................. 216

CREATE.................................. 218

REMOVE.................................. 222

REDESIGNATE............................. 226

UPDATE.................................. 229

SWAP$CONTACTS........................... 235

TRANSLATE............................... 238

OWN$SHIP$UPDATE......................... 240

ORIGIN.................................. 245

WIND.................................... 246

SCALE................................... 248

GET$SAFE$RNG............................ 249

INPUT$TIME.............................. 251

3. CPA$MODULE............................ 254

CONV$CONTACT$TIME....................... 255

CPA$TIME$CONV........................... 257

CONTACT$CRS$SPD......................... 259

CPA$CALCULATION......................... 261

GET$CPA................................. 274

4. DISPLAY$CMDS.......................... 277

CONV$LAT$LONG........................... 279

DISPLAY$DESIG........................... 281

DISPLAY$TYPE............................ 282

DISPLAY$CLASS........................... 283

DISPLAY$LAT$LONG........................ 284

DISPLAY$XY.............................. 286

DISPLAY$CRS$BRG......................... 287

DISPLAY$SPD............................. 288

DISPLAY$RANGE.............................. 289

DISPLAY$TIME............................... 290

DISPLAY$ORIGIN............................. 292

DISPLAY$SCALE.............................. 293

DISPLAY$OWN$SHIP........................... 294

DISPLAY$CONTACT$INFO....................... 296

DISPLAY$SYSTEM............................. 304

DISPLAY$SAFE$RNG........................... 306

DISPLAY$WIND............................... 307

DISPLAY$UPDATE$TIME........................ 309

5. COMMANDS................................... 310

PRINT$ERROR$MSG............................ 314

CHECK$YES$NO............................... 315

CHECK$FP$VALUE............................. 316

CHECK$INPUT................................ 317

GET$DEGREES................................ 318

GET$MINUTES................................ 320

GET$SIGN................................... 321

FP$FORMAT.................................. 322

RANGE$FORMAT............................... 325

LAT$LONG$FORMAT............................ 327

GET$TIME$ZONE.............................. 329

GET$LAT.................................... 331

GET$LONG................................... 333

GET$COURSE$BRG............................. 335

GET$SPEED.................................. 337

GET$RANGE.................................. 339

GET$DESIG.............................. 342

GET$TYPE............................... 344

GET$KIND............................... 346

GET$SCALE.............................. 348

6. PLASMA$MODULE........................ 350

SET$WINDOW............................. 353

CLEAR$STRUCTURES....................... 354

DRAW$FRIENDLY$SYMBOL................... 355

DRAW$UNK$HOS$SYMBOL.................... 357

DRAW$OWN$SHIP$SYMBOL................... 358

CHECK$PLASMA........................... 359

NORMALIZE.............................. 361

PUT$OS$CENTER.......................... 363

PUT$CONTACT$CENTER..................... 364

FIXED$REORIENTATION.................... 365

PLASMA$REDESIG......................... 367

PLASMA$DELETE.......................... 368

PLASMA$CONTACT......................... 369

PLASMA$OS.............................. 371

DRAW$EVERYTHING........................ 373

DISPLAY$PLASMA$SCALE................... 375

REORIENT$PS............................ 376

7. CRT.................................. 380

CRT$MASTER$CLEAR....................... 382

SET$LOW$HOME........................... 383

CLEAR$LOW$SCREEN....................... 384

SET$HIGH$HOME.......................... 385

PUT$SPACE.............................. 386

PUT$TAB................................ 387

PUT$FS................................. 388

PUT$LF................................. 389

START$PROT$FIELD....................... 390

START$BLINK............................ 391

STOP$PROT$FIELD........................ 392

INTERP................................. 393

INIT$HIGH$SCREEN....................... 395

PRINT$TIME$ZONE........................ 397

PRINT$TIME............................. 398

PRINT$LAT$LONG......................... 399

PRINT$COURSE........................... 400

PRINT$SPEED............................ 401

PRINT$CONTACTS......................... 402

PRINT$MODE............................. 403

PRINT$CONTACT$INFO..................... 404

8. FLTASCII............................ 405

ASCII$TO$FLOAT......................... 408

FRAC$TO$ASCII.......................... 411

FLOAT$TO$ASCII......................... 413

9. FLOATING$POINT...................... 417

INIT$FP................................ 420

ADJUST$OP.............................. 421

ADJUST1$OP............................. 422

ADJUST2$OP............................. 423

VAL$RESULT............................. 424

VAL$RESULT$1 425

VAL$RESULT$2 426

COMPARE 427

FLOAT$MSG$ERROR 429

CHECK 431

MUL 432

DIV 433

EDIV 434

FMUL 435

FDIV 436

FADD 437

FSUB 438

FSQR 439

FLTDS 440

FIXSD 441

FSQRT 442

FCMPR 443

FZTST 445

EXCH 447

COS$SIN 448

ARC$TAN 453

10. TIME 457

CLOCK 459

INITIATE$TIME 461

INITIATE$CLOCK 464

ACTUAL$TIME 465

11. PLASMA$PRIMITIVES 466

SET$STATUS$PLASMA.............................. 468
PLASMA$WRITE.................................. 469
CLEAR$PLASMA.................................. 470
PLASMA$WRITE$VECTOR........................... 471
PLASMA$PRINT$STRING........................... 472
INITIALIZE$PLASMA............................. 473
SET$VECTOR.................................... 474
START$VECTOR$SOLID............................ 475
STOP$VECTOR$SOLID............................. 476
START$VECTOR$DASH............................. 477
STOP$VECTOR$DASH.............................. 478
GRAPHIC$DESIG................................. 479
12. BASICS...................................... 481
CRT$WRITE..................................... 482
CRT$PRINT$STRING.............................. 483
CRT$READ...................................... 484
CRT$TRY$READ.................................. 485
ECHO$CRT...................................... 486
SEND$SUB...................................... 487
SEND$CR....................................... 488
SEND$LF....................................... 489
SEND$CRLF..................................... 490
SEND$BEL...................................... 491
SEND$BS....................................... 492
SEND$SPACE.................................... 493
BYTE$CHAR..................................... 494
ADDRESS$CHAR.................................. 495

BYTE$TO$ASCII.............................. 496

GET$BYTE.................................. 497

GET$ADDRESS............................... 499

GET$STRING................................ 501

PUT$NUMBER$BUFFER......................... 503

C. LISTINGS:

```
/***   EXTER:   ***/

DECLARE
   TIME$BUFFER(6) BYTE EXTERNAL,
   RES$TABLE(8) BYTE EXTERNAL,
   (MILI$SEC,DUMMY$SEC,SECONDS,MINUTES,HOURS,DAY,SEC$TIME) BYTE EXTERNAL,
   TIME$STEP ADDRESS EXTERNAL,
   MEBAS ADDRESS EXTERNAL;

CRT$WRITE:
   PROCEDURE (CHAR) EXTERNAL;
   DECLARE CHAR BYTE; END;

CRT$PRINT$STRING:
   PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS; END;

CRT$READ:
   PROCEDURE BYTE EXTERNAL;
   END;

CRT$TRY$READ:
   PROCEDURE BYTE EXTERNAL;
   END;

ECHO$CRT:
   PROCEDURE BYTE EXTERNAL;
   END;

SEND$SUB:
   PROCEDURE EXTERNAL;
```

```
    END;

SEND$CR:
    PROCEDURE EXTERNAL;
    END;

SEND$LF:
    PROCEDURE EXTERNAL;
    END;

SEND$CRLF:
    PROCEDURE EXTERNAL;
    END;

SEND$BEL:
    PROCEDURE EXTERNAL;
    END;

SEND$BS:
    PROCEDURE EXTERNAL;
    END;

SEND$SPACE:
    PROCEDURE (NUM) EXTERNAL;
    DECLARE NUM BYTE;  END;

BYTE$CHAR:
    PROCEDURE (CHAR) EXTERNAL;
    DECLARE CHAR BYTE;  END;

ADDRESS$CHAR:
```

```
    PROCEDURE (CHAR) EXTERNAL;
    DECLARE CHAR ADDRESS; END;

BYTE$TO$ASCII:
    PROCEDURE (A,B,C) EXTERNAL;
    DECLARE (A,B,C) ADDRESS; END;

GET$BYTE:
    PROCEDURE (A) BYTE EXTERNAL;
    DECLARE A BYTE; END;

GET$ADDRESS:
    PROCEDURE (A) ADDRESS EXTERNAL;
    DECLARE A BYTE; END;

GET$STRING:
    PROCEDURE (A,B) EXTERNAL;
    DECLARE A ADDRESS, B BYTE; END;

PUT$NUMBER$BUFFER:
    PROCEDURE (A,B) EXTERNAL;
    DECLARE A BYTE, B ADDRESS; END;

INIT$FP:
    PROCEDURE EXTERNAL;
    END;

MUL:
    PROCEDURE (A,B,C) EXTERNAL;
    DECLARE (A,B,C) ADDRESS; END;
```

```
DIV:
   PROCEDURE (A,B,C,D) EXTERNAL;
   DECLARE (A,B,C,D) ADDRESS;  END;

EDIV:
   PROCEDURE (A,B,C,D) EXTERNAL;
   DECLARE (A,B,C,D) ADDRESS;  END;

FMUL:
   PROCEDURE (A,B,C) EXTERNAL;
   DECLARE (A,B,C) ADDRESS;  END;

FDIV:
   PROCEDURE (A,B,C) EXTERNAL;
   DECLARE (A,B,C) ADDRESS;  END;

FADD:
   PROCEDURE (A,B,C) EXTERNAL;
   DECLARE (A,B,C) ADDRESS;  END;

FSUB:
   PROCEDURE (A,B,C) EXTERNAL;
   DECLARE (A,B,C) ADDRESS;  END;

FSQR:
   PROCEDURE (A,B) EXTERNAL;
   DECLARE (A,B) ADDRESS;  END;

FSQRT:
   PROCEDURE (A,B) EXTERNAL;
   DECLARE (A,B) ADDRESS;  END;
```

```
FLTDS:
   PROCEDURE (A, B) EXTERNAL;
   DECLARE (A, B) ADDRESS;  END;

FIXSD:
   PROCEDURE (A, B) EXTERNAL;
   DECLARE (A, B) ADDRESS;  END;

FCMPR:
   PROCEDURE (A, B, C) BYTE EXTERNAL;
   DECLARE (A, B, C) ADDRESS;  END;

FZTST:
   PROCEDURE (A, B) BYTE EXTERNAL;
   DECLARE (A, B) ADDRESS;  END;

EXCH:
   PROCEDURE (A, B) EXTERNAL;
   DECLARE (A, B) ADDRESS;  END;

COS$SIN:
   PROCEDURE (A, B, C) EXTERNAL;
   DECLARE (A, B, C) ADDRESS;  END;

ARC$TAN:
   PROCEDURE (A, B, C) EXTERNAL;
   DECLARE (A, B, C) ADDRESS;  END;

ASCII$TO$FLOAT:
   PROCEDURE (A, N, B) EXTERNAL;
```

```
   DECLARE (A,B) ADDRESS, N BYTE; END;

FLOAT$TO$ASCII:
   PROCEDURE (A,B,C) EXTERNAL;
   DECLARE (A,B,C) ADDRESS; END;

INITIATE$TIME:
   PROCEDURE EXTERNAL;
   END;

INITIATE$CLOCK:
   PROCEDURE EXTERNAL;
   END;

ACTUAL$TIME:
   PROCEDURE EXTERNAL;
   END;

CRT$MASTER$CLEAR:
   PROCEDURE EXTERNAL;
   END;

SET$LOW$HOME:
   PROCEDURE EXTERNAL;
   END;

CLEAR$LOW$SCREEN:
   PROCEDURE EXTERNAL;
   END;

SET$HIGH$HOME:
```

```
   PROCEDURE EXTERNAL;
   END;

INIT$HIGH$SCREEN:
   PROCEDURE EXTERNAL;
   END;

START$BLINK:
   PROCEDURE EXTERNAL;
   END;

PRINT$TIME$ZONE:
   PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS; END;

PRINT$TIME:
   PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS; END;

PRINT$LAT$LONG:
   PROCEDURE (A,B) EXTERNAL;
   DECLARE (A,B) ADDRESS; END;

PRINT$COURSE:
   PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS; END;

PRINT$SPEED:
   PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS; END;
```

```
PRINT$CONTACTS:
   PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS;  END;

PRINT$MODE:
   PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS;  END;

PRINT$CONTACT$INFO:
   PROCEDURE (A,B) EXTERNAL;
   DECLARE A BYTE,  B ADDRESS;  END;

CHECK$YES$NO:
   PROCEDURE BYTE EXTERNAL;
   END;

CHECK$FP$VALUE:
   PROCEDURE (A,B) BYTE EXTERNAL;
   DECLARE (A,B) ADDRESS;  END;

CHECK$INPUT:
   PROCEDURE BYTE EXTERNAL;
   END;

GET$DEGREES:
   PROCEDURE (A,B) EXTERNAL;
   DECLARE A BYTE)  B ADDRESS;  END;

GET$MINUTES:
   PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS;  END;
```

```
GET$SIGN:
   PROCEDURE (A,B) BYTE EXTERNAL;
   DECLARE (A,B) BYTE; END;

FP$FORMAT:
   PROCEDURE (A,B,C,D) BYTE  EXTERNAL;
   DECLARE (A,B) ADDRESS, (C,D) BYTE; END;

RANGE$FORMAT:
   PROCEDURE (A,B) EXTERNAL;
   DECLARE (A,B) ADDRESS; END;

LAT$LONG$FORMAT:
   PROCEDURE (A,B,C) EXTERNAL;
   DECLARE (A,B) ADDRESS, C BYTE; END;

GET$TIME$ZONE:
   PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS; END;

GET$LAT:
   PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS; END;

GET$LONG:
   PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS; END;

GET$COURSE$BRG:
   PROCEDURE (A,B) EXTERNAL;
```

```
   DECLARE A BYTE, B ADDRESS; END;

GET$SPEED:
   PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS; END;

GET$RANGE:
   PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS; END;

GET$DESIG:
   PROCEDURE ADDRESS EXTERNAL;
   END;

GET$TYPE:
   PROCEDURE BYTE EXTERNAL;
   END;

GET$KIND:
   PROCEDURE BYTE EXTERNAL;
   END;

GET$SCALE:
   PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS; END;
```

```
/***   EXTER$ONE:   ***/

DECLARE LIT LITERALLY 'LITERALLY',
        DCL LIT 'DECLARE';

DCL SYSTEM STRUCTURE
          (LAT (4) BYTE,
           LONG (4) BYTE,
           SCALE (4) BYTE,
           WIND$DIR (4) BYTE,
           WIND$SPD (4) BYTE,
           NUM$ZONE (5) BYTE,
           CONTACT$KIND (3) BYTE,
           NUMCTS BYTE ) EXTERNAL,

    OWN$SHIP$INFO STRUCTURE
                 (LAT (4) BYTE,
                  LONG (4) BYTE,
                  POINTER BYTE,
                  FLAG BYTE) EXTERNAL,

    OWN$SHIP (30) STRUCTURE
                 (X (4) BYTE,
                  Y (4) BYTE,
                  TIME (3) BYTE,
                  CRS (4) BYTE,
                  SPD (4) BYTE) EXTERNAL,

    CONTACT$INFO (15) STRUCTURE
                     (DESIG ADDRESS,
                      TYPE BYTE,
```

```
                      KIND BYTE,
                      CRS$FLAG BYTE,
                      SPD$FLAG BYTE,
                      OS$POINTER BYTE,
                      POINTER BYTE,
                      FLAG BYTE) EXTERNAL,

    CONTACT$POSI (225) STRUCTURE
                      (X (4) BYTE,
                       Y (4) BYTE,
                       TIME (3) BYTE,
                       CRS (4) BYTE,
                       SPD (4) BYTE,
                       BRG (4) BYTE,
                       RNG (4) BYTE) EXTERNAL;

DCL CONTACT$DISPLAY (6) BYTE EXTERNAL;


DCL
    LAT$STRING (9) BYTE EXTERNAL,
    LONG$STRING (9) BYTE EXTERNAL,
    CRS$STRING (6) BYTE EXTERNAL,
    SPD$STRING (5) BYTE EXTERNAL,
    CONTACTS$STRING (8) BYTE EXTERNAL,
    CONTACT$INFO$STRING (44) BYTE EXTERNAL;


DE$HASH:
   PROCEDURE (A,B) EXTERNAL;
   DCL (A,B) ADDRESS; END;
```

```
CHECK$GO$KEY:
    PROCEDURE EXTERNAL;
    END;

CONV$MIN$RAD:
    PROCEDURE (A,B) EXTERNAL;
    DCL (A,B) ADDRESS; END;

DISPLAY$KIND:
    PROCEDURE EXTERNAL;
    END;

CHECK$DESIG:
    PROCEDURE (A) BYTE EXTERNAL;
    DCL A ADDRESS; END;

CONV$RAD$MIN:
    PROCEDURE (A,B) EXTERNAL;
    DCL (A,B) ADDRESS; END;

CONV$XY:
    PROCEDURE (A,B,C,D) EXTERNAL;
    DCL (A,B,C,D) ADDRESS; END;

CONV$REL$XY:
    PROCEDURE (A,B,C,D) EXTERNAL;
    DCL (A,B,C,D) ADDRESS; END;

GET$SYSTEM$PARAMETERS:
    PROCEDURE EXTERNAL;
```

```
    END;

DISPLAY$CONTACT:
    PROCEDURE (A,B) EXTERNAL;
    DCL (A,B) BYTE;  END;

CREATE:
    PROCEDURE EXTERNAL;
    END;

REMOVE:
    PROCEDURE EXTERNAL;
    END;

REDESIGNATE:
    PROCEDURE EXTERNAL;
    END;

UPDATE:
    PROCEDURE EXTERNAL;
    END;

SWAP$CONTACTS:
    PROCEDURE EXTERNAL;
    END;

OWN$SHIP$UPDATE:
    PROCEDURE EXTERNAL;
    END;

ORIGIN:
```

```
   PROCEDURE EXTERNAL;
   END;

WIND:
   PROCEDURE EXTERNAL;
   END;

SCALE:
   PROCEDURE EXTERNAL;
   END;

INPUT$TIME:
   PROCEDURE BYTE EXTERNAL;
   END;
```

```
/***    EXTER$TWO:    ***/


SET$STATUS$PLASMA:
   PROCEDURE (A) EXTERNAL;
   DECLARE A BYTE;  END;

PLASMA$WRITE:
   PROCEDURE (A) EXTERNAL;
   DECLARE A BYTE;  END;

CLEAR$PLASMA:
   PROCEDURE EXTERNAL;
   END;

PLASMA$WRITE$VECTOR:
   PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS;  END;

PLASMA$PRINT$STRING:
   PROCEDURE (A, B, C) EXTERNAL;
   DECLARE (A, B) BYTE, C ADDRESS;  END;

INITIALIZE$PLASMA:
   PROCEDURE EXTERNAL;
   END;

SET$VECTOR:
   PROCEDURE (A, B, C) EXTERNAL;
   DECLARE (A, B, C) ADDRESS;  END;
```

```
START$VECTOR$SOLID:
   PROCEDURE (A, B) EXTERNAL;
   DECLARE (A, B) ADDRESS; END;

STOP$VECTOR$SOLID:
   PROCEDURE (A, B) EXTERNAL;
   DECLARE (A, B) ADDRESS; END;

START$VECTOR$DASH:
   PROCEDURE (A, B) EXTERNAL;
   DECLARE (A, B) ADDRESS; END;

STOP$VECTOR$DASH:
   PROCEDURE (A, B) EXTERNAL;
   DECLARE (A, B) ADDRESS; END;

GRAPHIC$DESIG:
   PROCEDURE (A, B, C) EXTERNAL;
   DECLARE (A, B, C) ADDRESS; END;
```

```
MAIN#MODULE: DO;
```

```
/***     EXTERNALS:    ***/


$NOLIST

$INCLUDE (:F1:EXTER.SRC)

$INCLUDE (:F1:EXTER1.SRC)

$LIST


CONV$LAT$LONG:
   PROCEDURE (A,B,C,D) EXTERNAL;
   DCL (A,B,C,D) ADDRESS;  END;

DISPLAY$WIND:
   PROCEDURE EXTERNAL;
   END;

DISPLAY$CONTACT$INFO:
   PROCEDURE EXTERNAL;
   END;

DISPLAY$ORIGIN:
   PROCEDURE EXTERNAL;
   END;

DISPLAY$SCALE:
   PROCEDURE EXTERNAL;
   END;
```

```
DISPLAY$OWN$SHIP:
    PROCEDURE EXTERNAL;
    END;

DISPLAY$SAFE$RNG:
    PROCEDURE EXTERNAL;
    END;

DISPLAY$SYSTEM:
    PROCEDURE EXTERNAL;
    END;

DISPLAY$UPDATE$TIME:
    PROCEDURE (A) EXTERNAL;
    DECLARE A BYTE; END;

GET$SAFE$RNG:
    PROCEDURE EXTERNAL;
    END;

CLEAR$PLASMA:
    PROCEDURE EXTERNAL;
    END;

INITIALIZE$PLASMA:
    PROCEDURE EXTERNAL;
    END;


SET$WINDOW:
```

```
    PROCEDURE EXTERNAL;
    END;

CLEAR$STRUCTURES:
    PROCEDURE EXTERNAL;
    END;

PUT$OS$CENTER:
    PROCEDURE EXTERNAL;
    END;

PLASMA$OS:
    PROCEDURE EXTERNAL;
    END;

DRAW$EVERYTHING:
    PROCEDURE EXTERNAL;
    END;

DISPLAY$PLASMA$SCALE:
    PROCEDURE EXTERNAL;
    END;

REORIENT$PS:
    PROCEDURE EXTERNAL;
    END;
```

```
/***   DECLARATIONS:   ***/

DCL TRUE LIT '0FFH',
    FALSE LIT '00H',
    FOREVER LIT 'WHILE TRUE',
    PROMPT LIT '025H',
    DISPLAY$UPPER$LIMIT LIT '01AH',
    INPUT$LOWER$LIMIT LIT '02DH',
    INPUT$UPPER$LIMIT LIT '03AH';


DCL DISPLAY (*) BYTE DATA ('DISPLAY$$'),
    INPUT   (*) BYTE DATA (' INPUT $$');


DCL TIME$LIMIT BYTE,
    TEMP BYTE,
    WIND$FLAG BYTE,
    COMMAND BYTE;
```

```
/***********************************************************************
*
* NO$WIND:
*    THIS PROCEDURE IS USED TO TELL THE OPERATOR THAT NO WIND INFORMATION
*    EXISTS IN THE SYSTEM.
*
***********************************************************************/
NO$WIND: PROCEDURE;
   DCL MSG (*) BYTE DATA
            ('NO WIND INFORMATION AVAILABLE. $$');

   CALL CRT$PRINT$STRING(.MSG);
   CALL SEND$CRLF;
   CALL SEND$CRLF;
   CALL CHECK$GO$KEY;
   CALL CLEAR$LOW$SCREEN;
   END NO$WIND;
```

```
/********************************************************************************
*
* NO$CONTACT:
*   THIS PROCEDURE IS USED TO TELL THE OPERATOR THAT THE SYSTEM IS NOT CUR-
*   RENTLY MAINTAINING ANY CONTACT.
*
********************************************************************************/
NO$CONTACT: PROCEDURE;
   DCL MSG (*) BYTE DATA
           ('NO CONTACTS ARE BEING MAINTAINED BY THE SYSTEM. $$');

   CALL CRT$PRINT$STRING(.MSG);
   CALL SEND$CRLF;
   CALL SEND$CRLF;
   CALL CHECK$GO$KEY;
   CALL CLEAR$LOW$SCREEN;
   END NO$CONTACT;
```

```
/***************************************************************************
*
* NOT$ENOUGH$CONTACTS:
*   THIS PROCEDURE IS USED TO TELL THE OPERATOR THAT THE NUMBER OF CONTACTS
*   PRESENTLY AT THE SYSTEM, IS NOT ENOUGH TO ALLOW SWAPING OF CONTACTS NOT
*   IN THE SCREEN.
*
***************************************************************************/
NOT$ENOUGH$CONTACTS: PROCEDURE;
   DCL MSG (*) BYTE DATA
           ('ALL CONTACTS IN THE SYSTEM ARE ALREADY DISPLAYED. $$');

   CALL CRT$PRINT$STRING(.MSG);
   CALL SEND$CRLF;
   CALL SEND$CRLF;
   CALL CHECK$GO$KEY;
   CALL CLEAR$LOW$SCREEN;
   END NOT$ENOUGH$CONTACTS;
```

```
/*****************************************************************************
*
* TOO$MANY$CONTACTS:
*   THIS PROCEDURE IS USED TO TELL THE OPERATOR THAT THE SYSTEM CAN NOT ACCEPT
*   ANY MORE NEW CONTACTS.
*
******************************************************************************/
TOO$MANY$CONTACTS: PROCEDURE;
   DCL MSG (*) BYTE DATA
           ('SYSTEM ALREADY MAINTAINS 15 CONTACTS. $$');

   CALL CRT$PRINT$STRING(.MSG);
   CALL SEND$CRLF;
   CALL SEND$CRLF;
   CALL CHECK$GO$KEY;
   CALL CLEAR$LOW$SCREEN;
   END TOO$MANY$CONTACTS;
```

```
/**********************************************************************************************
*
* MOVE$OWN$SHIP:
*   THIS PROCEDURE IS EXECUTED EACH TIME A PREDETERMINED PERIOD OF TIME ELAP-
*   SES.  IT IS USED TO CALCULATE THE MOVEMENTS OF THE SHIP DURING THAT PERIOD.
*
**********************************************************************************************/
MOVE$OWN$SHIP: PROCEDURE PUBLIC;
   DCL DELTA$X (4) BYTE,
       DELTA$Y (4) BYTE,
       COS (4) BYTE,
       SIN (4) BYTE,
       TIME (4) BYTE,
       TIME$FLOAT (4) BYTE,
       SPEED (4) BYTE,
       COURSE (4) BYTE,
       DISTANCE (4) BYTE,
       S ADDRESS,
       (I, POINTER, LAST$INFO, H, M) BYTE;

   DCL FP$3600 (4) BYTE DATA (00H, 00H, 61H, 45H),          /* 3600.00 */
       DEG$TO$RAD (4) BYTE DATA (035H, 0FAH, 08EH, 03CH);   /* 0.0174532925 */

      /* SAVE TIME OF CALL */
   H, M, S = 00H;
   S = TIME$STEP;
   TIME$STEP = 00H;
   TIME(1), TIME(2), TIME(3) = 00H;
   TIME(0) = LOW(S);
   IF S > 255
      THEN TIME(1) = HIGH(S);
```

```
    /* FIND INTERVAL OF TIME PAST */
DO WHILE S >= 60;
    S = S - 60;
    M = M + 1;
    END;
    /* CONVERT TIME TO FP FORMAT */
CALL FLTDS(.TIME, .TIME$FLOAT);
    /* CONVERT SPEED IN KNOTS INTO MILES/SECONDS */
POINTER = OWN$SHIP$INFO.POINTER;
CALL FDIV(.OWN$SHIP(POINTER).SPD, .FP$3600, .SPEED);
    /* CONVERT COURSE TO ANGLE IN RADIANS */
CALL FMUL(.OWN$SHIP(POINTER).CRS, .DEG$TO$RAD, .COURSE);
    /* GET SINE AND COSINE VALUES */
CALL COS$SIN(.COURSE, .COS, .SIN);
    /* FIND VARIATIONS IN X AND Y PARAMETERS */
CALL FMUL(.TIME$FLOAT, .SPEED, .DISTANCE);
CALL FMUL(.DISTANCE, .SIN, .DELTA$X);
CALL FMUL(.DISTANCE, .COS, .DELTA$Y);
    /* UPDATE OWN SHIP VALUES */
OWN$SHIP$INFO.POINTER = OWN$SHIP$INFO.POINTER + 1;
IF OWN$SHIP$INFO.POINTER = 30
THEN DO;
       OWN$SHIP$INFO.POINTER = 0;
       OWN$SHIP$INFO.FLAG = TRUE;
       END;
LAST$INFO = POINTER;
POINTER = OWN$SHIP$INFO.POINTER;
OWN$SHIP(POINTER).TIME(0) = OWN$SHIP(LAST$INFO).TIME(0) + H;
OWN$SHIP(POINTER).TIME(1) = OWN$SHIP(LAST$INFO).TIME(1) + M;
OWN$SHIP(POINTER).TIME(2) = OWN$SHIP(LAST$INFO).TIME(2) + S;
DO WHILE OWN$SHIP(POINTER).TIME(2) >= 60;
```

```
        OWN$SHIP(POINTER).TIME(2) = OWN$SHIP(POINTER).TIME(2) - 60;
        OWN$SHIP(POINTER).TIME(1) = OWN$SHIP(POINTER).TIME(1) + 1;
        END;
    DO WHILE OWN$SHIP(POINTER).TIME(1) >= 60;
        OWN$SHIP(POINTER).TIME(1) = OWN$SHIP(POINTER).TIME(1) - 60;
        OWN$SHIP(POINTER).TIME(0) = OWN$SHIP(POINTER).TIME(0) + 1;
        END;
    IF OWN$SHIP(POINTER).TIME(0) >= 24
        THEN OWN$SHIP(POINTER).TIME(0) = OWN$SHIP(POINTER).TIME(0) - 24;
    CALL FADD(.OWN$SHIP(LAST$INFO).X, .DELTA$X, .OWN$SHIP(POINTER).X);
    CALL FADD(.OWN$SHIP(LAST$INFO).Y, .DELTA$Y, .OWN$SHIP(POINTER).Y);
    CALL CONV$LAT$LONG(.OWN$SHIP(POINTER).X, .OWN$SHIP(POINTER).Y,
                         .OWN$SHIP$INFO.LAT, .OWN$SHIP$INFO.LONG);
    DO I = 0 TO 3;
        OWN$SHIP(POINTER).CRS(I) = OWN$SHIP(LAST$INFO).CRS(I);
        OWN$SHIP(POINTER).SPD(I) = OWN$SHIP(LAST$INFO).SPD(I);
        END;
        /* DISPLAY ACTUAL VALUES. */
    CALL LAT$LONG$FORMAT(.OWN$SHIP$INFO.LAT, .LAT$STRING, 0);
    CALL LAT$LONG$FORMAT(.OWN$SHIP$INFO.LONG, .LONG$STRING, 1);
    CALL PRINT$LAT$LONG(.LAT$STRING, .LONG$STRING);
        /* DRAW NEW POSITION IN PLASMA DISPLAY. */
    CALL PLASMA$OS;
        /* ALL DONE. RETURN */
    END MOVE$OWN$SHIP;
```

```
/***   EXECUTIVE:   ***/


EXECUTIVE: DO;


    TIME$STEP = 00H;
    TIME$LIMIT = 180;
    WIND$FLAG = FALSE;
    CALL CLEAR$STRUCTURES;
    CALL INITIALIZE$PLASMA;
    CALL GET$SYSTEM$PARAMETERS;
    CALL SET$WINDOW;
    CALL CLEAR$PLASMA;
    CALL DISPLAY$PLASMA$SCALE;
    CALL PUT$OS$CENTER;
    CALL PLASMA$OS;


    DO FOREVER;
       IF SEC$TIME          /* A SECOND HAS ELAPSED.  DISPLAY ACTUAL TIME */
       THEN DO;
            SEC$TIME = FALSE;
            CALL ACTUAL$TIME;
            CALL PRINT$TIME(.TIME$BUFFER);
            CALL CRT$WRITE(PROMPT);
            END;

       IF TIME$STEP >= TIME$LIMIT
          THEN CALL MOVE$OWN$SHIP;      /* TIME TO UPDATE OWN SHIP POSITION */
```

```
COMMAND = CRT$TRY$READ;        /* CHECK FOR INPUT FROM KEYBOARD */

IF (COMMAND < DISPLAY$UPPER$LIMIT) AND
   (COMMAND <> 0)
                      /* CHECK FOR DISPLAY TYPE COMMANDS */
THEN DO;
     CALL PRINT$MODE(.DISPLAY);
     DO CASE COMMAND;
        ;                 /* CASE 00H */

        DO;               /* CASE 01H */
          IF WIND$FLAG
             THEN CALL DISPLAY$WIND;
             ELSE CALL NO$WIND;
        END;

        ;                 /* CASE 02H */

        ;                 /* CASE 03H */

        DO;               /* CASE 04H */
          CALL DISPLAY$SAFE$RNG;
        END;

        ;                 /* CASE 05H */

        DO;               /* CASE 06H */
          CALL DISPLAY$UPDATE$TIME(TIME$LIMIT);
        END;
```

```
;             /* CASE 07H */

;             /* CASE 08H */

;             /* CASE 09H */

;             /* CASE 0AH */

DO;           /* CASE 0BH */
  IF SYSTEM.NUMCTS > 0
     THEN CALL DISPLAY$CONTACT$INFO;
     ELSE CALL NO$CONTACT;
END;

DO;           /* CASE 0CH */
  CALL DISPLAY$ORIGIN;
END;

;             /* CASE 0DH */

DO;           /* CASE 0EH */
  CALL DISPLAY$SCALE;
END;

DO;           /* CASE 0FH */
  CALL DISPLAY$OWN$SHIP;
END;

;             /* CASE 10H */

;             /* CASE 11H */
```

```
        ;             /* CASE 12H */

        ;             /* CASE 13H */

        ;             /* CASE 14H */

        ;             /* CASE 15H */

        ;             /* CASE 16H */

        ;             /* CASE 17H */

        ;             /* CASE 18H */

        DO;           /* CASE 19H */
          IF SYSTEM.NUMCTS > 0
            THEN CALL DISPLAY$SYSTEM;
            ELSE CALL NO$CONTACT;
        END;

        END;          /* END CASE */
   CALL PRINT$MODE(.INPUT);
   END;               /* END THEN DO */

IF (COMMAND > INPUT$LOWER$LIMIT) AND (COMMAND < INPUT$UPPER$LIMIT)
THEN DO;              /* INPUT COMMAND DETECTED */
     DO CASE (COMMAND - 2EH);
        DO;           /* CASE 2EH */
          CALL REORIENT$PS;
        END;
```

```
DO;           /* CASE 2FH */
  IF SYSTEM.NUMCTS > 0
     THEN CALL UPDATE;
     ELSE CALL NO$CONTACT;
END;

DO;             /* CASE 30H */
  CALL GET$SAFE$RNG;
END;

DO;             /* CASE 31H */
  IF SYSTEM.NUMCTS > 0
     THEN CALL REDESIGNATE;
     ELSE CALL NO$CONTACT;
END;

DO;             /* CASE 32H */
  IF(TEMP:= INPUT$TIME) <> 0
     THEN TIME$LIMIT = TEMP;
END;

DO;             /* CASE 33H */
  CALL SCALE;
END;

DO;             /* CASE 34H */
  IF SYSTEM.NUMCTS > 0
     THEN CALL REMOVE;
     ELSE CALL NO$CONTACT;
END;
```

```
DO;                  /* CASE 35H */
   IF (SYSTEM.NUMCTS > 6)
       THEN CALL SWAP$CONTACTS;
       ELSE DO;
             IF SYSTEM.NUMCTS <> 0
                 THEN CALL NOT$ENOUGH$CONTACTS;
                 ELSE CALL NO$CONTACT;
             END;
END;

DO;                  /* CASE 36H */
   CALL WIND;
   WIND$FLAG = TRUE;
END;

DO;                  /* CASE 37H */
   IF SYSTEM.NUMCTS < 15
   THEN DO;
          CALL CREATE;
          SYSTEM.NUMCTS = SYSTEM.NUMCTS + 1;
          END;
   ELSE CALL TOO$MANY$CONTACTS;
END;

DO;                  /* CASE 38H */
   CALL OWN$SHIP$UPDATE;
END;

DO;                  /* CASE 39H */
   CALL ORIGIN;
```

```
              END;

              END;            /* END CASE */
        END;                  /* END THEN DO */
      END;                    /* END DO FOREVER */


   END EXECUTIVE;


END MAIN$MODULE;
```

```
EXECUTIVE$CMDS: DO;


$NOLIST

$INCLUDE (:F1:EXTER.SRC)

$LIST
```

```
/***    EXTERNALS:    ***/


DECLARE LIT LITERALLY 'LITERALLY',
        DCL LIT 'DECLARE';


GET$CPA:
   PROCEDURE (A, B) BYTE EXTERNAL;
   DCL A BYTE, B ADDRESS; END;

PLASMA$REDESIG:
   PROCEDURE (INDEX) EXTERNAL;
   DCL INDEX BYTE; END;

PLASMA$DELETE:
   PROCEDURE (INDEX) EXTERNAL;
   DCL INDEX BYTE; END;

PLASMA$CONTACT:
   PROCEDURE (INDEX) EXTERNAL;
   DCL INDEX BYTE; END;

CLEAR$STRUCTURES:
   PROCEDURE EXTERNAL;
   END;

SET$WINDOW:
   PROCEDURE EXTERNAL;
   END;
```

```
PUT#OS#CENTER:
    PROCEDURE EXTERNAL;
    END;

DRAW#EVERYTHING:
    PROCEDURE EXTERNAL;
    END;

PLASMA#OS:
    PROCEDURE EXTERNAL;
    END;

DISPLAY#PLASMA#SCALE:
    PROCEDURE EXTERNAL;
    END;

MOVE#OWN#SHIP:
    PROCEDURE EXTERNAL;
    END;
```

```
/***   DECLARATIONS:   ***/



DCL SAFE$RNG (4) BYTE PUBLIC INITIAL (08H, 3CH, 0CAH, 03CH);
               /* 0.0246868278 MILES OR FIFTY YARDS */


DCL SYSTEM STRUCTURE
          (LAT (4) BYTE,
           LONG (4) BYTE,
           SCALE (4) BYTE,
           WIND$DIR (4) BYTE,
           WIND$SPD (4) BYTE,
           NUM$ZONE (5) BYTE,
           CONTACT$KIND (3) BYTE,
           NUMCTS BYTE ) PUBLIC,

    OWN$SHIP$INFO STRUCTURE
                 (LAT (4) BYTE,
                  LONG (4) BYTE,
                  POINTER BYTE,
                  FLAG BYTE) PUBLIC,

    OWN$SHIP (30) STRUCTURE
                 (X (4) BYTE,
                  Y (4) BYTE,
                  TIME (3) BYTE,
                  CRS (4) BYTE,
                  SPD (4) BYTE) PUBLIC,
```

```
    CONTACT$INFO (15) STRUCTURE
                     (DESIG ADDRESS,
                      TYPE BYTE,
                      KIND BYTE,
                      CRS$FLAG BYTE,
                      SPD$FLAG BYTE,
                      OS$POINTER BYTE,
                      POINTER BYTE,
                      FLAG BYTE) PUBLIC,

    CONTACT$POSI (225) STRUCTURE
                     (X (4) BYTE,
                      Y (4) BYTE,
                      TIME (3) BYTE,
                      CRS (4) BYTE,
                      SPD (4) BYTE,
                      BRG (4) BYTE,
                      RNG (4) BYTE) PUBLIC;

DCL CONTACT$DISPLAY (6) BYTE PUBLIC;

DCL LAT$STRING (9) BYTE PUBLIC,
    LONG$STRING (9) BYTE PUBLIC,
    CRS$STRING (6) BYTE PUBLIC,
    SPD$STRING (5) BYTE PUBLIC,
    CONTACTS$STRING (8) BYTE PUBLIC,
    CONTACT$INFO$STRING (44) BYTE PUBLIC,
    INPUT$MODE (*) BYTE DATA (' INPUT $$'),
    DISPLAY$MODE (*) BYTE DATA ('DISPLAY$$');
```

```
DCL FP$MIN$TO$RAD (4) BYTE DATA (98H,82H,98H,39H),    /* 0.00029089 */
    FP$2 (4) BYTE DATA (00H,00H,00H,40H);             /* 2.0 */

DCL PROMPT LIT '25H';       /* PROMPT CHARACTER */


DCL
    MSG$0 (*) BYTE DATA ('PRESS THE ''GO'' KEY TO CONTINUE:$$'),
    MSG$1 (*) BYTE DATA ('DO YOU NEED TO UPDATE$$'),
    MSG$2 (*) BYTE DATA ('COURSE?    (Y/N) $$'),
    MSG$3 (*) BYTE DATA ('SPEED?     (Y/N) $$'),
    MSG$4 (*) BYTE DATA ('LATITUDE?  (Y/N) $$'),
    MSG$5 (*) BYTE DATA ('LONGITUDE? (Y/N) $$'),
    MSG$6 (*) BYTE DATA ('BEARING?   (Y/N) $$'),
    MSG$7 (*) BYTE DATA ('RANGE?     (Y/N) $$'),
    MSG$8 (*) BYTE DATA ('DESIG?     (Y/N) $$'),
    MSG$9 (*) BYTE DATA ('TYPE?      (Y/N) $$'),
    MSG$A (*) BYTE DATA ('CLASS?     (Y/N) $$'),
    BLANK (*) BYTE DATA ('                    $$');


DCL
    TITLE$0 (*) BYTE DATA
            ('                           NEW CONTACT INITIALIZATION.$$'),
    TITLE$1 (*) BYTE DATA
            ('                               CONTACT REMOVAL.$$'),
    TITLE$2 (*) BYTE DATA
            ('                           CONTACT REDESIGNATION.$$'),
    TITLE$3 (*) BYTE DATA
            ('                                CONTACT UPDATE.$$'),
```

```
   TITLE$4 (*) BYTE DATA
           ('               OWN SHIP DATA UPDATING.$$'),
   TITLE$5 (*) BYTE DATA
           ('       CHANGE OF CONTACTS BEING DISPLAYED.$$'),
   TITLE$6 (*) BYTE DATA
           ('               CHANGE OF TIME PARAMETERS.$$'),
   TITLE$9 (*) BYTE DATA
           ('       COORDINATE GRID ORIGIN MODIFICATION.$$'),
   TITLE$A (*) BYTE DATA
           ('            CHANGE OF WIND INFORMATION.$$'),
   TITLE$B (*) BYTE DATA
           ('            GRAPHICS SCALE MODIFICATION.$$'),
   TITLE$C (*) BYTE DATA
           ('                PICTURE REORIENTATION.$$'),
   TITLE$D (*) BYTE DATA
           ('            CHANGE OF SAFE C.P.A.  RANGE$$');
```

```
/****************************************************************************
*
* DE$HASH:
*   THIS PROCEDURE IS USED TO GET A PAIR OF ASCII CHARACTERS, REPRESENTING
*   A CONTACT'S DESIGNATION, FROM AN ADDRESS VALUE.  SEE 'GET$DESIG'.
*
* PARAMETERS:
*   - A. - ADDRESS VALUE CONTAINING THE CODE TO BE 'DEHASHED' INTO TWO
*     ASCII CHARACTERS.
*   - B. - POINTER TO A MEMORY LOCATION IN WHICH THE STRING IS DESIRED TO BE
*     PLACED.
*
****************************************************************************/
DE$HASH: PROCEDURE (A,B) PUBLIC;
   DCL (A,B) ADDRESS,
       CHAR BASED B BYTE;
   CHAR = A / 100;
   B = B + 1;
   CHAR = A MOD 100;
   END DE$HASH;
```

```
/*********************************************************************************************
*
* CHECK$GO$KEY:
*   THIS PROCEDURE IS USED TO CHECK IF THE 'GO' KEY IS PRESSED.
*
*************************************************************************************************/
CHECK$GO$KEY: PROCEDURE PUBLIC;
    DCL CHAR BYTE;
    CALL CRT$PRINT$STRING(.MSG$0);
    CALL SEND$BEL;
    CHAR = CRT$READ;
    DO WHILE CHAR <> 02CH;
        CALL SEND$BEL;
        CHAR = CRT$READ;
        END;
    END CHECK$GO$KEY;
```

```
/*******************************************************************************
*
* DISPLAY$KIND:
*   THIS PROCEDURE IS USED TO DISPLAY THE INFORMATION ABOUT THE KIND OF
*   CONTACTS MAINTAINED BY THE SYSTEM.
*
********************************************************************************/
DISPLAY$KIND: PROCEDURE PUBLIC;
   DCL I BYTE;
   DO I = 0 TO 2;
      CONTACTS$STRING(2 * I) = SYSTEM.CONTACT$KIND(I) / 10 + 30H;
      CONTACTS$STRING((2 * I) + 1) = SYSTEM.CONTACT$KIND(I) MOD 10 + 30H;
      END;
   CONTACTS$STRING(6), CONTACTS$STRING(7) = '$';
   CALL PRINT$CONTACTS(.CONTACTS$STRING);
   END DISPLAY$KIND;
```

```
/*****************************************************************************
*
* CHECK$DESIG:
*   THIS PROCEDURE IS USED TO USED TO DETECT THE PRESENCE OF A GIVEN CONTACT
*   IN THE SYSTEM.
*
* PARAMETERS:
*   - A. - ADDRESS VARIABLE THAT CONTAINS THE 'HASHED' VALUE OF THE CONTACT'S
*     DESIGNATION DESIRED TO BE CHECKED.
*
* USAGE:
*   TYPED PROCEDURE.  A VALUE INDICATING THE RELATIVE POSITION OF THE CONTACT
*   IS RETURNED IF FOUND.  OTHERWISE A VALUE OF 0FFH IS RETURNED.
*
*****************************************************************************/
CHECK$DESIG: PROCEDURE (A) BYTE PUBLIC;
   DCL A ADDRESS,
       I BYTE;
   DO I = 0 TO 14;
      IF CONTACT$INFO(I).DESIG = A THEN RETURN I;
      END;
   RETURN 0FFH;
   END CHECK$DESIG;
```

```
/******************************************************************************************
*
* CONV$MIN$RAD:
*   THIS PROCEDURE IS USED TO CONVERT A GIVEN ANGLE, IN MINUTES, TO RADIANS.
*
* PARAMETERS:
*   - A. - POINTER TO A MEMORY LOCATION IN WHICH THE FLOATING POINT REPRESEN-
*       TATION OF AN ANGLE IN MINUTES, IS LOCATED.
*   - B. - POINTER TO A MEMORY LOCATION IN WHICH THE VALUE IN RADIANS IS DESIRED
*       TO BE PLACED.
*
*****************************************************************************************/
CONV$MIN$RAD: PROCEDURE (A,B) PUBLIC;
   DCL (A,B) ADDRESS,
       MIN BASED A (4) BYTE,
       RAD BASED B (4) BYTE;
   CALL FMUL(.MIN,.FP$MIN$TO$RAD,.RAD);
   END CONV$MIN$RAD;
```

```
/*******************************************************************************
*
* CONV$RAD$MIN:
*   THIS PROCEDURE IS USED TO CONVERT A GIVEN ANGLE, IN RADIANS, TO MINUTES.
*
* PARAMETERS:
*   - A. - POINTER TO A MEMORY LOCATION IN WHICH THE FLOATING POINT REPRESEN-
*     TATION OF AN ANGLE IN RADIANS, IS LOCATED.
*   - B. - POINTER TO A MEMORY LOCATION IN WHICH THE VALUE IN MINUTES IS DESI-
*     RED TO BE PLACED.
*
*******************************************************************************/
CONV$RAD$MIN: PROCEDURE (A,B) PUBLIC;
   DCL (A,B) ADDRESS,
       RAD BASED A (4) BYTE,
       MIN BASED B (4) BYTE;
   CALL FDIV(.RAD,.FP$MIN$TO$RAD,.MIN);
   END CONV$RAD$MIN;
```

```
/****************************************************************************
*
* CONV$XY:
*   THIS PROCEDURE IS USED TO CONVERT GIVEN VALUES OF LATITUDE AND LONGITUDE
*   INTO 'X,Y' COORDINATES.
*
* PARAMETERS:
*   - A. - POINTER TO A MEMORY LOCATION IN WHICH THE FP VALUE OF LATITUDE IS
*     LOCATED.
*   - B. - POINTER TO A MEMORY LOCATION IN WHICH THE FP VALUE OF LONGITUDE IS
*     LOCATED.
*   - C. - POINTER TO A MEMORY LOCATION IN WHICH THE FP VALUE REPRESENTING
*     'X' IS DESIRED TO BE PLACED.
*   - D. - POINTER TO A MEMORY LOCATION IN WHICH THE FP VALUE REPRESENTING
*     'Y' IS DESIRED TO BE PLACED.
*
****************************************************************************/
CONV$XY: PROCEDURE (A,B,C,D) PUBLIC;
   DCL (A,B,C,D) ADDRESS,
       LAT BASED A (4) BYTE,
       LONG BASED B (4) BYTE,
       X BASED C (4) BYTE,
       Y BASED D (4) BYTE,
       MEAN$LAT (4) BYTE,
       COS$MEAN$LAT (4) BYTE,
       SIN$MEAN$LAT (4) BYTE;
   CALL FADD(.SYSTEM.LAT, .LAT, .MEAN$LAT);
   CALL FDIV(.MEAN$LAT, .FP$2, .MEAN$LAT);
   CALL CONV$MIN$RAD(.MEAN$LAT, .MEAN$LAT);
   CALL COS$SIN(.MEAN$LAT, .COS$MEAN$LAT, .SIN$MEAN$LAT);
   CALL FSUB(.LONG, .SYSTEM.LONG, .X);
```

```
CALL FMUL(.X, .COS$MEAN$LAT, .X);
CALL FSUB(.LAT, .SYSTEM.LAT, .Y);
END CONV$XY;
```

```
/****************************************************************************
*
* CONV$REL$XY:
*   THIS PROCEDURE IS USED TO OBTAIN VALUES OF 'X,Y' COORDINATES FOR A POINT
*   TO WHICH A RANGE AND BEARING ARE GIVEN.
*
* PARAMETERS:
*   - A. - POINTER TO A MEMORY LOCATION IN WHICH THE FP VALUE OF THE BEARING
*     IS LOCATED.
*   - B. - POINTER TO A MEMORY LOCATION IN WHICH THE FP VALUE OF THE RANGE
*     IS LOCATED.
*   - C. - POINTER TO A MEMORY LOCATION IN WHICH THE FP VALUE REPRESENTING
*     'X' IS DESIRED TO BE PLACED.
*   - D. - POINTER TO A MEMORY LOCATION IN WHICH THE FP VALUE REPRESENTING
*     'Y' IS DESIRED TO BE PLACED.
*
******************************************************************************/
CONV$REL$XY: PROCEDURE (A,B,C,D) PUBLIC;
   DCL (A,B,C,D) ADDRESS,
        BRG BASED A (4) BYTE,
        RNG BASED B (4) BYTE,
        DELTA$X BASED C (4) BYTE,
        DELTA$Y BASED D (4) BYTE,
        COS (4) BYTE, SIN (4) BYTE,
        ANGLE (4) BYTE,
        (I, TEMP) BYTE;
   DCL DEG$TO$RAD (4) BYTE DATA (035H, 0FAH, 08EH, 03CH);   /* 0.0174532925 */
   TEMP = OWN$SHIP$INFO.POINTER;
   DO I = 0 TO 3;
      ANGLE(I) = BRG(I);
      END;
```

```
CALL FMUL(.ANGLE, .DEG$TO$RAD, .ANGLE);
CALL COS$SIN(.ANGLE, .COS, .SIN);
CALL FMUL(.RNG, .SIN, .DELTA$X);
CALL FADD(.OWN$SHIP(TEMP).X, .DELTA$X, .DELTA$X);
CALL FMUL(.RNG, .COS, .DELTA$Y);
CALL FADD(.OWN$SHIP(TEMP).Y, .DELTA$Y, .DELTA$Y);
END CONV$REL$XY;
```

```
/***************************************************************************
*
* INIT$STRUCTURES:
*   THIS PROCEDURE IS USED TO INITIALIZE ALL STRUCTURES TO 0.
*
*****************************************************************************/
INIT$STRUCTURES: PROCEDURE;
   DCL (A,I) ADDRESS,
       TEMP BASED A BYTE;
   A = .SYSTEM;
   DO I = 0 TO 28;
      TEMP = 0;
      A = A + 1;
      END;
   A = .OWN$SHIP$INFO;
   DO I = 0 TO 9;
      TEMP = 0;
      A = A + 1;
      END;
   A = .OWN$SHIP;
   DO I = 0 TO 569;
      TEMP = 0;
      A = A + 1;
      END;
   A = .CONTACT$INFO;
   DO I = 0 TO 134;
      TEMP = 0;
      A = A + 1;
      END;
   A = .CONTACT$POSI;
   DO I = 0 TO 6074;
```

```
        TEMP = 0;
        A = A + 1;
        END;
    A = .CONTACT$DISPLAY;
    DO I = 0 TO 5;
        TEMP = 0FFH;
        A = A + 1;
        END;
    END INIT$STRUCTURES;
```

```
/****************************************************************************
*
* GET$SYSTEM$PARAMETERS:
*   THIS PROCEDURE IS USED BY THE EXECUTIVE TO INITIALIZE THE SYSTEM.
*
****************************************************************************/
GET$SYSTEM$PARAMETERS: PROCEDURE PUBLIC;
   DCL MSG0 (*) BYTE DATA
       ('                             SYSTEM INITIALIZATION$$'),
       MSG1 (*) BYTE DATA ('TIME INITIALIZATION:$$'),
       MSG2 (*) BYTE DATA ('COORDINATE GRID ORIGIN INITIALIZATION:$$'),
       MSG3 (*) BYTE DATA ('OWN SHIP INITIAL DATA:$$'),
       MSG4 (*) BYTE DATA ('INITIAL GRAPHICS SCALE:$$'),
       MSG5 (*) BYTE DATA ('PRESS THE ''GO'' KEY TO START THE SYSTEM.$$');


   DCL CHAR BYTE;

   CALL CRT$MASTER$CLEAR;
   CALL INIT$FP;
   CALL INIT$STRUCTURES;
   CALL INIT$HIGH$SCREEN;
   CALL SET$LOW$HOME;
   CALL CRT$PRINT$STRING(.MSG0);
   CALL SEND$CRLF;
   CALL CRT$PRINT$STRING(.MSG1);
   CALL SEND$CRLF;
   CALL GET$TIME$ZONE(.SYSTEM.NUM$ZONE);
   CALL CRT$PRINT$STRING(.MSG0);
   CALL SEND$CRLF;
   CALL CRT$PRINT$STRING(.MSG1);
```

```
CALL SEND$CRLF;
CALL INITIATE$TIME;
CALL CLEAR$LOW$SCREEN;
CALL CRT$PRINT$STRING(.MSG0);
CALL SEND$CRLF;
CALL CRT$PRINT$STRING(.MSG2);
CALL SEND$CRLF;
CALL GET$LAT(.SYSTEM.LAT);
CALL CRT$PRINT$STRING(.MSG0);
CALL SEND$CRLF;
CALL CRT$PRINT$STRING(.MSG2);
CALL SEND$CRLF;
CALL GET$LONG(.SYSTEM.LONG);
CALL CRT$PRINT$STRING(.MSG0);
CALL SEND$CRLF;
CALL CRT$PRINT$STRING(.MSG3);
CALL SEND$CRLF;
OWN$SHIP(0).TIME(0) = HOURS;
OWN$SHIP(0).TIME(1) = MINUTES;
OWN$SHIP(0).TIME(2) = SECONDS;
CALL GET$LAT(.OWN$SHIP$INFO.LAT);
CALL CRT$PRINT$STRING(.MSG0);
CALL SEND$CRLF;
CALL CRT$PRINT$STRING(.MSG3);
CALL SEND$CRLF;
CALL GET$LONG(.OWN$SHIP$INFO.LONG);
CALL CRT$PRINT$STRING(.MSG0);
CALL SEND$CRLF;
OWN$SHIP$INFO.POINTER = 0;
CALL CONV$XY(.OWN$SHIP$INFO.LAT, .OWN$SHIP$INFO.LONG, .OWN$SHIP(0).X,
             .OWN$SHIP(0).Y);
```

```
CALL CRT$PRINT$STRING(.MSG3);
CALL SEND$CRLF;
CALL GET$COURSE$BRG(0, .OWN$SHIP(0).CRS);
CALL CRT$PRINT$STRING(.MSG0);
CALL SEND$CRLF;
CALL CRT$PRINT$STRING(.MSG3);
CALL SEND$CRLF;
CALL GET$SPEED(.OWN$SHIP(0).SPD);
CALL CRT$PRINT$STRING(.MSG0);
CALL SEND$CRLF;
CALL CRT$PRINT$STRING(.MSG4);
CALL SEND$CRLF;
CALL GET$SCALE(.SYSTEM.SCALE);
CALL CRT$PRINT$STRING(.MSG0);
CALL SEND$CRLF;
CALL START$BLINK;
CALL CRT$PRINT$STRING(.MSG5);
CALL CRT$WRITE(18H);                /*   STOP BLINK MODE   */
CHAR = 0;
DO WHILE CHAR <> 2CH;               /*   WAIT FOR ORDER TO START   */
   IF CHAR <> 2CH THEN CALL SEND$BEL;
   CHAR = CRT$READ;
   END;
CALL INITIATE$CLOCK;
CALL CLEAR$LOW$SCREEN;
CALL ACTUAL$TIME;
CALL PRINT$TIME(.TIME$BUFFER);
CALL PRINT$TIME$ZONE(.SYSTEM.NUM$ZONE);
CALL LAT$LONG$FORMAT(.OWN$SHIP$INFO.LAT, .LAT$STRING, 0);
CALL LAT$LONG$FORMAT(.OWN$SHIP$INFO.LONG, .LONG$STRING, 1);
CALL PRINT$LAT$LONG(.LAT$STRING, .LONG$STRING);
```

```
CHAR = FP$FORMAT(.OWN$SHIP(0).CRS, .CRS$STRING, 3, 1);
CALL PRINT$COURSE(.CRS$STRING);
CHAR = FP$FORMAT(.OWN$SHIP(0).SPD, .SPD$STRING, 2, 1);
CALL PRINT$SPEED(.SPD$STRING);
DO CHAR = 0 TO 5;
   CONTACTS$STRING(CHAR) = '0';
   END;
CONTACTS$STRING(6), CONTACTS$STRING(7) = '$';
CALL PRINT$CONTACTS(.CONTACTS$STRING);
CALL PRINT$MODE(.INPUT$MODE);
CONTACT$INFO$STRING(42), CONTACT$INFO$STRING(43) = '$';
CALL CRT$WRITE(PROMPT);
END GET$SYSTEM$PARAMETERS;
```

```
/*************************************************************************************
*
* DISPLAY$CONTACT:
*   THIS PROCEDURE IS USED TO DISPLAY ALL THE AVAILABLE INFORMATION ABOUT
*   A GIVEN CONTACT, ACCORDING TO THE FORMAT PRESENT AT THE CRT.
*
* PARAMETERS:
*   - ROW. - INDICATES IN WHICH DISPLAY ROW TO PUT THE INFORMATION.
*   - INDEX. - SHOWS THE RELATIVE POSITION OF THE CONTACT IN THE DATA STRUC-
*     TURES BEING USED.
*
*************************************************************************************/
DISPLAY$CONTACT: PROCEDURE (ROW, INDEX) PUBLIC;
   DCL KIND (*) BYTE DATA ('FRIHOSUNK'),
       (ROW, INDEX, I, J, TEMP, TEST) BYTE;
   CONTACT$DISPLAY(ROW) = INDEX;
   ROW = ROW + 1;
   CALL DE$HASH (CONTACT$INFO(INDEX).DESIG, .CONTACT$INFO$STRING(0));
   CONTACT$INFO$STRING(2), CONTACT$INFO$STRING(3) = 'S';
   IF CONTACT$INFO(INDEX).TYPE = 0
      THEN CONTACT$INFO$STRING(2) = ' ';
   J = CONTACT$INFO(INDEX).KIND;
   DO I = 0 TO 2;
      CONTACT$INFO$STRING(4 + I) = KIND((3 * J) + I);
      END;
   J = CONTACT$INFO(INDEX).POINTER;
   CONTACT$INFO$STRING(7) = CONTACT$POSI(J).TIME(0) / 10 + 30H;
   CONTACT$INFO$STRING(8) = CONTACT$POSI(J).TIME(0) MOD 10 + 30H;
   CONTACT$INFO$STRING(9) = CONTACT$POSI(J).TIME(1) / 10 + 30H;
   CONTACT$INFO$STRING(10) = CONTACT$POSI(J).TIME(1) MOD 10 + 30H;
   TEMP = FP$FORMAT(.CONTACT$POSI(J).BRG, .CONTACT$INFO$STRING(11), 3, 1);
```

```
CALL RANGE$FORMAT(.CONTACT$POSI(J).RNG, .CONTACT$INFO$STRING(15));
TEMP = GET$CPA(INDEX, .CONTACT$INFO$STRING(21));
IF TEMP = 0
THEN DO;
   IF CONTACT$INFO(INDEX).CRS$FLAG
   THEN DO;
       TEST = FP$FORMAT(.CONTACT$POSI(J).CRS,
                        .CONTACT$INFO$STRING(21), 3, 1);
      END;
   ELSE DO;
      DO I = 21 TO 24;
         CONTACT$INFO$STRING(I) = ' ';
         END;
      END;
   IF CONTACT$INFO(INDEX).SPD$FLAG
   THEN DO;
       TEST = FP$FORMAT(.CONTACT$POSI(J).SPD,
                        .CONTACT$INFO$STRING(25), 2, 1);
      END;
   ELSE DO;
      DO I = 25 TO 27;
         CONTACT$INFO$STRING(I) = ' ';
         END;
      END;
   DO I = 28 TO 41;
      CONTACT$INFO$STRING(I) = ' ';
      END;
   END;
CONTACT$INFO$STRING(42), CONTACT$INFO$STRING(43) = '$';
CALL PRINT$CONTACT$INFO(ROW, .CONTACT$INFO$STRING);
END DISPLAY$CONTACT;
```

```
/*****************************************************************************
*
* CREATE:
*    THIS PROCEDURE IS USED TO OBTAIN ALL PERTINENT INFORMATION ABOUT A NEW
*    CONTACT.
*
* USAGE:
*    UNTYPED PROCEDURE.  THIS PROCEDURE SHOULD BE CALLED AFTER IT HAS BEEN
*    DETERMINED THAT A CONTACT CAN BE ACCEPTED IN THE SYSTEM, AND THAT THE
*    NUMBER OF CONTACTS HAS BEEN UPDATED.
*
****************************************************************************/
CREATE: PROCEDURE PUBLIC;
   DCL STR1 (*) BYTE DATA ('ARE THE FOLLOWING VALUES KNOWN?$$'),
       ARRAY (2) BYTE,
       A ADDRESS,
       (OK, TEMP, I, J, INDEX, H, M, S) BYTE;
   H = HOURS;     /* TO SAVE THE TIME OF CALL */
   M = MINUTES;
   S = SECONDS;
      /* UPDATE POSITION OF OWN SHIP */
   CALL MOVE$OWN$SHIP;
      /* GET INITIAL CONTACT VALUES */
   OK = 0;
   DO WHILE OK = 0;
      CALL CRT$PRINT$STRING(.TITLE$0);
      CALL SEND$CRLF;
      CALL CRT$PRINT$STRING(.STR1);
      CALL SEND$CRLF;
      CALL CRT$PRINT$STRING(.MSG$2);
      ARRAY(0) = CHECK$YES$NO;
```

```
      CALL SEND$CRLF;
      CALL CRT$PRINT$STRING(.MSG$3);
      ARRAY(1) = CHECK$YES$NO;
      CALL SEND$CRLF;
      OK = CHECK$INPUT;
      CALL CLEAR$LOW$SCREEN;
      END;
   TEMP, I = 0;
   DO WHILE TEMP = 0;
      IF CONTACT$INFO(I).DESIG = 0
      THEN DO;
         TEMP = 1;
         INDEX = I;
         END;
      I = I + 1;
      END;
   OK = 0;
   DO WHILE OK = 0;      /* GET DESIG */
      CALL CRT$PRINT$STRING(.TITLE$0);
      CALL SEND$CRLF;
      A, CONTACT$INFO(INDEX).DESIG = GET$DESIG;
      DO I = 0 TO 14;
         IF (I <> INDEX) AND (A = CONTACT$INFO(I).DESIG)
         THEN DO;
            CALL CRT$PRINT$STRING(.('DESIGNATION ALREADY IN USE.$$'));
            CALL SEND$CRLF;
            CALL CHECK$GO$KEY;
            OK = 0;
            CALL CLEAR$LOW$SCREEN;
            GO TO L;
            END;
```

```
        OK = 1;
        END;
L: END;
CALL CRT$PRINT$STRING(.TITLE$0);
CALL SEND$CRLF;
CONTACT$INFO(INDEX).TYPE = GET$TYPE;    /* GET TYPE */
CALL CRT$PRINT$STRING(.TITLE$0);
CALL SEND$CRLF;
TEMP, CONTACT$INFO(INDEX).KIND = GET$KIND;    /* GET KIND */
SYSTEM.CONTACT$KIND(TEMP) = SYSTEM.CONTACT$KIND(TEMP) + 1;
CALL CRT$PRINT$STRING(.TITLE$0);
CALL SEND$CRLF;
J, CONTACT$INFO(INDEX).POINTER = 15*INDEX;
CONTACT$INFO(INDEX).OS$POINTER = 0FFH;
CONTACT$INFO(INDEX).FLAG = 0;
CONTACT$INFO(INDEX).CRS$FLAG = 0;
CONTACT$INFO(INDEX).SPD$FLAG = 0;
CONTACT$POSI(J).TIME(0) = H;
CONTACT$POSI(J).TIME(1) = M;
CONTACT$POSI(J).TIME(2) = S;
CALL GET$COURSE$BRG(1, .CONTACT$POSI(J).BRG);
CALL CRT$PRINT$STRING(.TITLE$0);
CALL SEND$CRLF;
CALL GET$RANGE(.CONTACT$POSI(J).RNG);
CALL CONV$REL$XY(.CONTACT$POSI(J).BRG, .CONTACT$POSI(J).RNG,
                 .CONTACT$POSI(J).X, .CONTACT$POSI(J).Y);
IF ARRAY(0)
THEN DO;
   CONTACT$INFO(INDEX).CRS$FLAG = 1;
   CALL CRT$PRINT$STRING(.TITLE$0);
   CALL SEND$CRLF;
```

```
      CALL GET$COURSE$BRG(0, .CONTACT$POSI(J).CRS);
      END;
   ELSE CONTACT$INFO(INDEX).CRS$FLAG = 0;
   IF ARRAY(1)
   THEN DO;
      CONTACT$INFO(INDEX).SPD$FLAG = 1;
      CALL CRT$PRINT$STRING(.TITLE$0);
      CALL SEND$CRLF;
      CALL GET$SPEED(.CONTACT$POSI(J).SPD);
      END;
   ELSE CONTACT$INFO(INDEX).SPD$FLAG = 0;
   I, OK = 0;
   DO WHILE (OK = 0) AND ( I <= LAST(CONTACT$DISPLAY));
      IF CONTACT$DISPLAY(I) = 0FFH
      THEN DO;
         OK = 1;
         CALL DISPLAY$CONTACT(I, INDEX);
         END;
      I = I + 1;
      END;
   CALL DISPLAY$KIND;
   CALL PLASMA$CONTACT(INDEX);
   END CREATE;
```

```
/*****************************************************************************
*
* REMOVE:
*    THIS PROCEDURE IS USED TO REMOVE A CONTACT FROM THE SYSTEM.
*
*****************************************************************************/
REMOVE: PROCEDURE PUBLIC;
    DCL DESIG ADDRESS,
        STRING(4) BYTE,
        (ROW, OK, I, CHAR, TEMP, CLASS, CHECK) BYTE;
    OK = 0;
    DO WHILE OK = 0;
        CALL CRT$PRINT$STRING(.TITLE$1);
        CALL SEND$CRLF;
        CALL CRT$PRINT$STRING(.
            ('ARE YOU SURE YOU WANT TO DELETE A CONTACT? (Y/N) $$'));
        IF ((CHAR:= CHECK$YES$NO) = 0 )      /* NOT SURE */
        THEN DO;
                CALL CLEAR$LOW$SCREEN;
                RETURN;
                END;
        CALL SEND$CRLF;
        DESIG = GET$DESIG;
        CALL DE$HASH(DESIG, .STRING);
        STRING(2), STRING(3) = '$';
        CALL CRT$PRINT$STRING(.TITLE$1);
        CALL SEND$CRLF;
        IF (TEMP:= CHECK$DESIG(DESIG)) <> 0FFH
        THEN DO;
                CALL CRT$PRINT$STRING(.('CONTACT TO BE DELETED: $$'));
                CALL CRT$PRINT$STRING(.STRING);
```

```
        OK = 1;
        END;
   ELSE DO;
        CALL CRT$PRINT$STRING(. ('DESIG NOT IN USE.  $$'));
        END;
   CALL SEND$CRLF;
   CALL SEND$CRLF;
   CALL CHECK$GO$KEY;
   CALL CLEAR$LOW$SCREEN;
   END;
CONTACT$INFO(TEMP).DESIG = 00H;
SYSTEM.NUMCTS = SYSTEM.NUMCTS - 1;
CLASS = CONTACT$INFO(TEMP).KIND;
SYSTEM.CONTACT$KIND(CLASS) = SYSTEM.CONTACT$KIND(CLASS) - 1;
ROW = 0FFH;
DO I = 0 TO 5;
   IF CONTACT$DISPLAY(I) = TEMP
   THEN DO;
        CONTACT$DISPLAY(I) = 0FFH;
        ROW = I;
        END;
   END;
CALL PLASMA$DELETE(TEMP);
IF (SYSTEM.NUMCTS > 5) AND (ROW <> 0FFH)
THEN DO;
     OK = 0;
     DO WHILE OK = 0;
        CALL CRT$PRINT$STRING(.
            ('ENTER THE DESIG OF A CONTACT DESIRED TO BE DISPLAYED: $$'));
        CALL SEND$CRLF;
        DESIG = GET$DESIG;
```

```
CALL DE$HASH(DESIG, .STRING);
STRING(2), STRING(3) = '$';
TEMP = CHECK$DESIG(DESIG);
IF TEMP = 0FFH
THEN DO;
     CALL CRT$PRINT$STRING(.('DESIG $$'));
     CALL CRT$PRINT$STRING(.STRING);
     CALL CRT$PRINT$STRING(.(' NOT IN USE. $$'));
     CALL SEND$CRLF;
     END;
ELSE DO;
     CHECK = 0;
     DO I = 0 TO LAST(CONTACT$DISPLAY);
        IF CONTACT$DISPLAY(I) = TEMP
        THEN DO;
           CALL CRT$PRINT$STRING(.('CONTACT ALREADY DISPLAYED. $$'));
           CALL SEND$CRLF;
           CHECK = 1;
           I = LAST(CONTACT$DISPLAY) + 2;
           END;
        END;
     IF CHECK = 0
     THEN DO;
        CALL CRT$PRINT$STRING(.('CONTACT $$'));
        CALL CRT$PRINT$STRING(.STRING);
        CALL CRT$PRINT$STRING(.(' WILL BE DISPLAYED. $$'));
        CALL SEND$CRLF;
        OK = 1;
        END;
     END;
CALL SEND$CRLF;
```

```
        CALL CHECK$GO$KEY;
        CALL CLEAR$LOW$SCREEN;
        END;
     IF TEMP <> 0FFH
     THEN DO;
          OK = 1;
          CALL DISPLAY$CONTACT(ROW, TEMP);
          END;
     END;
ELSE DO;
     IF ROW <> 0FFH
     THEN DO;
          DO I = 0 TO 41;
             CONTACT$INFO$STRING(I) = ' ';
             END;
          CONTACT$INFO$STRING(42), CONTACT$INFO$STRING(43) = '$';
          CALL PRINT$CONTACT$INFO(ROW + 1, .CONTACT$INFO$STRING);
          END;
     END;
CALL DISPLAY$KIND;
END REMOVE;
```

```
/*****************************************************************************
*
* REDESIGNATE:
*    THIS PROCEDURE IS USED TO CHANGE THE DESIG OF A CONTACT.
*
*******************************************************************************/
REDESIGNATE: PROCEDURE PUBLIC;
   DCL (NEW,OLD) ADDRESS,
       STR1 (4) BYTE,
       STR2 (4) BYTE,
       (TEMP, TEMP1, INDEX, I) BYTE;
   TEMP = 0FFH;
   DO WHILE TEMP = 0FFH;
      CALL CRT$PRINT$STRING(.TITLE$2);
      CALL SEND$CRLF;
      CALL CRT$PRINT$STRING(.('ENTER OLD DESIG AS REQUESTED:$$'));
      CALL SEND$CRLF;
      OLD = GET$DESIG;
      TEMP = CHECK$DESIG(OLD);
      IF TEMP = 0FFH
      THEN DO;
         CALL CRT$PRINT$STRING(.('DESIG NOT IN USE.$$'));
         CALL SEND$CRLF;
         CALL CHECK$GO$KEY;
         CALL CLEAR$LOW$SCREEN;
         END;
      END;
   TEMP1 = 0;
   DO WHILE TEMP1 <> 0FFH;
      CALL CRT$PRINT$STRING(.TITLE$2);
      CALL SEND$CRLF;
```

```
CALL CRT$PRINT$STRING(.('ENTER NEW DESIG AS REQUESTED:$$'));
CALL SEND$CRLF;
NEW = GET$DESIG;
TEMP1 = CHECK$DESIG(NEW);
IF TEMP1 <> 0FFH
THEN DO;
   CALL CRT$PRINT$STRING(.('DESIG ALREADY IN USE. $$'));
   CALL SEND$CRLF;
   END;
ELSE DO;
   STR1(2), STR1(3), STR2(2), STR2(3) = '$';
   CALL DE$HASH(OLD, .STR1);
   CALL DE$HASH(NEW, .STR2);
   CALL CRT$PRINT$STRING(.('CONTACT $$'));
   CALL CRT$PRINT$STRING(.STR1);
   CALL CRT$PRINT$STRING(.(' WILL BE CHANGED TO $$'));
   CALL CRT$PRINT$STRING(.STR2);
   CALL CRT$WRITE('.');
   CALL SEND$CRLF;
   CONTACT$INFO(TEMP1).DESIG = NEW;
   CALL SEND$CRLF;
   END;
CALL CHECK$GO$KEY;
CALL CLEAR$LOW$SCREEN;
IF TEMP1 = 0FFH
THEN DO;
   DO I = 0 TO LAST(CONTACT$DISPLAY);
      IF CONTACT$DISPLAY(I) = TEMP
      THEN DO;
         INDEX = CONTACT$DISPLAY(I);
         CONTACT$INFO(INDEX).DESIG = NEW;
```

```
               CALL PRINT$CONTACT$INFO(I + 1,  .STR2);
               END;
            END;
         END;
      END;
   CALL PLASMA$REDESIG(TEMP1);
   END REDESIGNATE;
```

```
/****************************************************************************
*
* UPDATE:
*   THIS PROCEDURE IS USED TO UPDATE INFORMATION ABOUT ANY CONTACT.
*
******************************************************************************/
UPDATE: PROCEDURE PUBLIC;
   DCL DESIG ADDRESS,
       ARRAY (6) BYTE,
       (LAST$INFO, OK, I, J, K, TEMP, KIND, OLD$KIND, TYPE, INDEX, COUNT, H, M, S) BYTE;
   H = HOURS;        /* SAVE TIME OF CALL */
   M = MINUTES;
   S = SECONDS;
      /* UPDATE OWN SHIP POSITION */
   CALL MOVE$OWN$SHIP;
      /* GET CONTACT VALUES */
   OK = 0FFH;
   DO WHILE OK = 0FFH;
      CALL CRT$PRINT$STRING(. TITLE$3);
      CALL SEND$CRLF;
      CALL CRT$PRINT$STRING(. ('ENTER CONTACT DESIG AS REQUESTED:$$'));
      CALL SEND$CRLF;
      DESIG = GET$DESIG;
      INDEX = CHECK$DESIG(DESIG);
      IF INDEX = 0FFH
      THEN DO;
         CALL CRT$PRINT$STRING(. ('DESIG NOT IN USE. $$'));
         CALL SEND$CRLF;
         CALL CHECK$GO$KEY;
         CALL CLEAR$LOW$SCREEN;
         END;
```

```
        OK = INDEX;
        END;
    OK = 0;
    DO WHILE OK = 0;
        CALL CRT$PRINT$STRING(.TITLE$3);
        CALL SEND$CRLF;
        CALL CRT$PRINT$STRING(.MSG$1);
        CALL SEND$CRLF;
        CALL CRT$PRINT$STRING(.MSG$9);
        ARRAY (0) = CHECK$YES$NO;
        CALL CRT$PRINT$STRING(.BLANK);
        CALL CRT$PRINT$STRING(.MSG$A);
        ARRAY (1) = CHECK$YES$NO;
        CALL SEND$CRLF;
        CALL CRT$PRINT$STRING(.MSG$6);
        ARRAY (2) = CHECK$YES$NO;
        CALL CRT$PRINT$STRING(.BLANK);
        CALL CRT$PRINT$STRING(.MSG$7);
        ARRAY (3) = CHECK$YES$NO;
        CALL SEND$CRLF;
        CALL CRT$PRINT$STRING(.MSG$2);
        ARRAY (4) = CHECK$YES$NO;
        CALL CRT$PRINT$STRING(.BLANK);
        CALL CRT$PRINT$STRING(.MSG$3);
        ARRAY (5) = CHECK$YES$NO;
        CALL SEND$CRLF;
        OK = CHECK$INPUT;
        CALL CLEAR$LOW$SCREEN;
        END;
    TEMP = 0;
    DO I = 0 TO LAST(ARRAY);
```

```
        IF ARRAY(I) THEN TEMP = 1;
        END;
    IF TEMP = 0 THEN RETURN;    /* NO INPUT IS DESIRED */
    J, LAST$INFO = CONTACT$INFO(INDEX).POINTER;
    IF ARRAY(2) OR ARRAY(3)
    THEN DO;
        CONTACT$INFO(INDEX).POINTER = CONTACT$INFO(INDEX).POINTER + 1;
        IF CONTACT$INFO(INDEX).POINTER = 15*INDEX + 15
        THEN DO;
            CONTACT$INFO(INDEX).POINTER = 15*INDEX;
            CONTACT$INFO(INDEX).FLAG = 0FFH;
            END;
        IF CONTACT$INFO(INDEX).OS$POINTER <> 0FFH
        THEN DO;
              IF CONTACT$INFO(INDEX).FLAG
              THEN DO;
                    TEMP = CONTACT$INFO(INDEX).OS$POINTER;
                    COUNT = 1;
                    DO WHILE TEMP <> CONTACT$INFO(INDEX).POINTER;
                        IF TEMP = (INDEX + 1)*15
                        THEN TEMP = INDEX*15;
                        ELSE TEMP = TEMP + 1;
                        COUNT = COUNT + 1;
                        END;
                    END;
              ELSE DO;
                    COUNT = CONTACT$INFO(INDEX).POINTER -
                             CONTACT$INFO(INDEX).OS$POINTER;
                    END;
              IF COUNT >= 5
              THEN CONTACT$INFO(INDEX).OS$POINTER = 0FFH;
```

```
      END;
    J = CONTACT$INFO(INDEX).POINTER;
    CONTACT$POSI(J).TIME(0) = H;
    CONTACT$POSI(J).TIME(1) = M;
    CONTACT$POSI(J).TIME(2) = S;
    END;
DO I = 0 TO LAST(ARRAY);
    IF ARRAY(I)
    THEN DO;
        CALL CRT$PRINT$STRING(.TITLE$3);
        CALL SEND$CRLF;
        DO CASE I;
            DO;
                CONTACT$INFO(INDEX).TYPE = GET$TYPE;
            END;
            DO;
                OLD$KIND = CONTACT$INFO(INDEX).KIND;
                KIND, CONTACT$INFO(INDEX).KIND = GET$KIND;
                SYSTEM.CONTACT$KIND(OLD$KIND) = SYSTEM.CONTACT$KIND(OLD$KIND) -1;
                SYSTEM.CONTACT$KIND(KIND) = SYSTEM.CONTACT$KIND(KIND) + 1;
            END;
            DO;
                CALL GET$COURSE$BRG(1, .CONTACT$POSI(J).BRG);
            END;
            DO;
                CALL GET$RANGE(.CONTACT$POSI(J).RNG);
            END;
            DO;
                CALL GET$COURSE$BRG(0, .CONTACT$POSI(J).CRS);
                CONTACT$INFO(INDEX).CRS$FLAG = 1;
            END;
```

```
        DO;
            CALL GET$SPEED(.CONTACT$POSI(J).SPD);
            CONTACT$INFO(INDEX).SPD$FLAG = 1;
        END;
        END;    /* END CASE */
    END;        /* IF THEN */
ELSE DO;
    IF ARRAY(2) OR ARRAY(3)
    THEN DO;
    DO CASE I;
        DO;
        END;
        DO;
        END;
        DO;
            IF (NOT ARRAY(2)) AND ARRAY(3)
            THEN DO;
                DO K = 0 TO 3;
                    CONTACT$POSI(J).BRG(K) = CONTACT$POSI(LAST$INFO).BRG(K);
                    END;
                END;
        END;
        DO;
            IF (NOT ARRAY(3)) AND ARRAY(2)
            THEN DO;
                DO K = 0 TO 3;
                    CONTACT$POSI(J).RNG(K) = CONTACT$POSI(LAST$INFO).RNG(K);
                    END;
                END;
        END;
        DO;
```

```
            DO K = 0 TO 3;
               CONTACT$POSI(J).CRS(K) = CONTACT$POSI(LAST$INFO).CRS(K);
               END;
         END;
         DO;
            DO K = 0 TO 3;
               CONTACT$POSI(J).SPD(K) = CONTACT$POSI(LAST$INFO).SPD(K);
               END;
         END;
         END;      /* END CASE */
         END;      /* IF THEN */
      END;         /* ELSE */
   END;            /* END DO */
IF ARRAY(2) OR ARRAY(3)
   THEN DO;
        CALL CONV$REL$XY(.CONTACT$POSI(J).BRG, .CONTACT$POSI(J).RNG,
                         .CONTACT$POSI(J).X,   .CONTACT$POSI(J).Y);
        CALL PLASMA$CONTACT(INDEX);
        END;
IF ARRAY (1)
   THEN CALL DISPLAY$KIND;
I, OK = 0;
DO WHILE (OK = 0) AND (I <= LAST(CONTACT$DISPLAY));
   IF CONTACT$DISPLAY(I) = INDEX
   THEN DO;
      OK = 1;
      CALL DISPLAY$CONTACT(I, INDEX);
      END;
   I = I + 1;
   END;
END UPDATE;
```

```
/*****************************************************************************************
*
* SWAP$CONTACTS:
*   THIS PROCEDURE IS USED TO SWAP ONE CONTACT BEING DISPLAYED WITH
*   ANOTHER WICH IS IN THE SYSTEM BUT NOT AT THE DISPLAY.
*
*****************************************************************************************/
SWAP$CONTACTS: PROCEDURE PUBLIC;
   DCL (CONTACT$IN, CONTACT$OUT) ADDRESS,
       STRING (4) BYTE,
       (TEMP, TEMP1, INDEX, I, J) BYTE;
   TEMP, TEMP1, J = 0FFH;
   DO WHILE TEMP = 0FFH;
      CALL CRT$PRINT$STRING(.TITLE$5);
      CALL SEND$CRLF;
      CALL CRT$PRINT$STRING(.('ENTER CONTACT TO BE OUT OF DISPLAY:$$'));
      CALL SEND$CRLF;
      CONTACT$OUT = GET$DESIG;
      TEMP = CHECK$DESIG(CONTACT$OUT);
      IF TEMP = 0FFH
      THEN DO;
           CALL CRT$PRINT$STRING(.('CONTACT NOT IN SYSTEM.$$'));
           CALL SEND$CRLF;
           CALL CHECK$GO$KEY;
           CALL CLEAR$LOW$SCREEN;
           END;
      ELSE DO;
           DO I = 0 TO LAST(CONTACT$DISPLAY);
              IF CONTACT$DISPLAY(I) = TEMP
              THEN DO;
                   J = I;
```

```
                I = 6;
                END;
           END;
        IF J = 0FFH
        THEN DO;
             TEMP = 0FFH;
             CALL CRT$PRINT$STRING(. ('CONTACT NOT AT DISPLAY. $$'));
             CALL SEND$CRLF;
             CALL CHECK$GO$KEY;
             CALL CLEAR$LOW$SCREEN;
             END;
        END;
   END;
DO WHILE TEMP1 = 0FFH;
   CALL CRT$PRINT$STRING(. TITLE$5);
   CALL SEND$CRLF;
   CALL CRT$PRINT$STRING(. ('ENTER CONTACT TO BE IN THE DISPLAY:$$'));
   CALL SEND$CRLF;
   CONTACT$IN = GET$DESIG;
   TEMP1 = CHECK$DESIG (CONTACT$IN);
   IF TEMP1 = 0FFH
   THEN DO;
        CALL CRT$PRINT$STRING(. ('CONTACT NOT IN SYSTEM. $$'));
        CALL SEND$CRLF;
        CALL CHECK$GO$KEY;
        CALL CLEAR$LOW$SCREEN;
        END;
   ELSE DO;
        DO I = 0 TO LAST (CONTACT$DISPLAY);
           IF CONTACT$DISPLAY(I) = TEMP1
           THEN DO;
```

```
                TEMP1 = 0FFH;
                CALL CRT$PRINT$STRING(.('CONTACT ALREADY DISPLAYED.$$'));
                CALL SEND$CRLF;
                CALL CHECK$GO$KEY;
                CALL CLEAR$LOW$SCREEN;
                I = LAST(CONTACT$DISPLAY) + 2;
                END;
           END;
        END;
   END;
CALL DISPLAY$CONTACT (J, TEMP1);
CALL CLEAR$LOW$SCREEN;
END SWAP$CONTACTS;
```

```
/*******************************************************************************
*
* TRANSLATE:
*   THIS PROCEDURE IS USED TO TRANSLATE ALL X/Y VALUES IN THE SYSTEM, BY GI-
*   VEN VALUES.
*
* PARAMETERS:
*   - A. - POINTER TO A FOUR BYTE VECTOR IN WHICH THE CHANGE IN X IS LOCATED.
*   - B. - POINTER TO A FOUR BYTE VECTOR IN WHICH THE CHANGE IN Y IS LOCATED.
*   - TEMP. - CAN HAVE TWO VALUES:
*              - 0: DO NOT CHANGE LAST POSITION OF OWN SHIP.
*              - 1: CHANGE ALL X/Y VALUES WITHOUT EXCEPTION.
*
*******************************************************************************/
TRANSLATE: PROCEDURE (A, B, TEMP);
   DCL (A, B) ADDRESS,
       X$DELTA BASED A (4) BYTE,
       Y$DELTA BASED B (4) BYTE,
       (I, J, P, NUM$PTS, TEMP, FLAG) BYTE;
   IF OWN$SHIP$INFO.FLAG THEN NUM$PTS = 29;
                         ELSE NUM$PTS = OWN$SHIP$INFO.POINTER;
   DO I = 0 TO NUM$PTS;
      FLAG = 0;
      DO WHILE ((TEMP = 1) OR ((TEMP = 0) AND (I <> OWN$SHIP$INFO.POINTER)))
               AND (FLAG = 0);
         CALL FADD(.OWN$SHIP(I).X, .X$DELTA, .OWN$SHIP(I).X);
         CALL FADD(.OWN$SHIP(I).Y, .Y$DELTA, .OWN$SHIP(I).Y);
         FLAG = 1;
         END;
      END;
   DO I = 0 TO 14;
```

```
   IF CONTACT$INFO(I).DESIG <> 0
   THEN DO;
        IF CONTACT$INFO(I).FLAG THEN NUM$PTS = 14;
                           ELSE NUM$PTS = CONTACT$INFO(I).POINTER MOD 15;
        DO J = 0 TO NUM$PTS;
           P = I*15 + J;
           CALL FADD(.CONTACT$POSI(P).X, .X$DELTA, .CONTACT$POSI(P).X);
           CALL FADD(.CONTACT$POSI(P).Y, .Y$DELTA, .CONTACT$POSI(P).Y);
           END;
        END;
    END;
END TRANSLATE;
```

```
/*************************************************************************************
*
* OWN$SHIP$UPDATE:
*   THIS PROCEDURE IS USED TO UPDATE THE INFORMATION ABOUT THE OWN SHIP.
*
*************************************************************************************/
OWN$SHIP$UPDATE: PROCEDURE PUBLIC;
   DCL ARRAY(4) BYTE,
       OLD$LAT (4) BYTE,
       OLD$LONG (4) BYTE,
       X$DELTA (4) BYTE,
       Y$DELTA (4) BYTE,
       (LAST$INFO, OK, I, J, K, TEMP, H, M, S) BYTE;
   H = HOURS;    /* SAVE TIME OF CALL */
   M = MINUTES;
   S = SECONDS;
      /* UPDATE OWN SHIP POSITION */
   CALL MOVE$OWN$SHIP;
      /* GET OWN SHIP VALUES */
   OK = 0;
   DO WHILE OK = 0;
      CALL CRT$PRINT$STRING(.TITLE$4);
      CALL SEND$CRLF;
      CALL CRT$PRINT$STRING(.MSG$1);
      CALL SEND$CRLF;
      CALL CRT$PRINT$STRING(.MSG$4);
      ARRAY(0) = CHECK$YES$NO;
      CALL CRT$PRINT$STRING(.BLANK);
      CALL CRT$PRINT$STRING(.MSG$5);
      ARRAY(1) = CHECK$YES$NO;
      CALL SEND$CRLF;
```

```
  CALL CRT$PRINT$STRING(.MSG$2);
  ARRAY(2) = CHECK$YES$NO;
  CALL CRT$PRINT$STRING(.BLANK);
  CALL CRT$PRINT$STRING(.MSG$3);
  ARRAY(3) = CHECK$YES$NO;
  CALL SEND$CRLF;
  OK = CHECK$INPUT;
  CALL CLEAR$LOW$SCREEN;
  END;
TEMP = 0;
DO I = 0 TO LAST(ARRAY);
  IF ARRAY(I) THEN TEMP = 1;
  END;
IF TEMP = 0 THEN RETURN;      /* NO INPUT IS DESIRED.  */
LAST$INFO = OWN$SHIP$INFO.POINTER;
OWN$SHIP$INFO.POINTER = OWN$SHIP$INFO.POINTER + 1;
IF OWN$SHIP$INFO.POINTER = 30
THEN DO;
  OWN$SHIP$INFO.POINTER = 0;
  OWN$SHIP$INFO.FLAG = 0FFH;
  END;
J = OWN$SHIP$INFO.POINTER;
OWN$SHIP(J).TIME(0) = H;
OWN$SHIP(J).TIME(1) = M;
OWN$SHIP(J).TIME(2) = S;
IF ARRAY(0) OR ARRAY(1)
THEN DO;
    DO I = 0 TO 3;
      OLD$LAT(I) = OWN$SHIP$INFO.LAT(I);
      OLD$LONG(I) = OWN$SHIP$INFO.LONG(I);
      END;
```

```
     END;
DO I = 0 TO LAST(ARRAY);
   IF ARRAY(I)
   THEN DO;
      CALL CRT$PRINT$STRING(.TITLE$4);
      CALL SEND$CRLF;
      DO CASE I;
         DO;
           CALL GET$LAT(.OWN$SHIP$INFO.LAT);
         END;
         DO;
           CALL GET$LONG(.OWN$SHIP$INFO.LONG);
         END;
         DO;
           CALL GET$COURSE$BRG(0, .OWN$SHIP(J).CRS);
           TEMP = FP$FORMAT(.OWN$SHIP(J).CRS, .CRS$STRING, 3, 1);
         END;
         DO;
           CALL GET$SPEED(.OWN$SHIP(J).SPD);
           TEMP = FP$FORMAT(.OWN$SHIP(J).SPD, .SPD$STRING, 2, 1);
         END;
         END;      /* CASE */
      END;         /* IF THEN */
   ELSE DO;
      DO CASE I;
         DO;
           DO K = 0 TO 3;
              OWN$SHIP(J).Y(K) = OWN$SHIP(LAST$INFO).Y(K);
              END;
         END;
         DO;
```

```
           DO K = 0 TO 3;
              OWN$SHIP(J).X(K) = OWN$SHIP(LAST$INFO).X(K);
              END;
         END;
         DO;
           DO K = 0 TO 3;
              OWN$SHIP(J).CRS(K) = OWN$SHIP(LAST$INFO).CRS(K);
              END;
         END;
         DO;
           DO K = 0 TO 3;
              OWN$SHIP(J).SPD(K) = OWN$SHIP(LAST$INFO).SPD(K);
              END;
         END;
         END;     /* CASE */
      END;        /* ELSE */
   END;           /* END DO */

IF ARRAY(2) OR ARRAY(3)
THEN DO;
     DO K = 0 TO 14;
        IF CONTACT$INFO(K).DESIG <> 00H
        THEN CONTACT$INFO(K).OS$POINTER = CONTACT$INFO(K).POINTER;
        END;
     END;
IF ARRAY(0) OR ARRAY(1)
THEN DO;
     CALL CONV$XY(.OWN$SHIP$INFO.LAT, .OWN$SHIP$INFO.LONG,
                  .OWN$SHIP(J).X, .OWN$SHIP(J).Y);
     CALL LAT$LONG$FORMAT(.OWN$SHIP$INFO.LAT, .LAT$STRING, 0);
     CALL LAT$LONG$FORMAT(.OWN$SHIP$INFO.LONG, .LONG$STRING, 1);
```

```
        CALL PRINT$LAT$LONG(.LAT$STRING, .LONG$STRING);
        CALL CONV$XY(.OLD$LAT, .OLD$LONG, .X$DELTA, .Y$DELTA);
        CALL FSUB(.OWN$SHIP(J).X, .X$DELTA, .X$DELTA);
        CALL FSUB(.OWN$SHIP(J).Y, .Y$DELTA, .Y$DELTA);
        CALL TRANSLATE(.X$DELTA, .Y$DELTA, 0);
        CALL CLEAR$STRUCTURES;
        CALL SET$WINDOW;
        CALL PUT$OS$CENTER;
        CALL DRAW$EVERYTHING;
        CALL DISPLAY$PLASMA$SCALE;
        END;
     IF ARRAY(2) THEN CALL PRINT$COURSE(.CRS$STRING);
     IF ARRAY(3) THEN CALL PRINT$SPEED(.SPD$STRING);
     END OWN$SHIP$UPDATE;
```

```
/***************************************************************************
*
* ORIGIN:
*   THIS PROCEDURE IS USED TO MODIFY THE INFORMATION ABOUT THE
*   COORDINATE GRID ORIGIN.
*
*****************************************************************************/
ORIGIN: PROCEDURE PUBLIC;
   DCL OLD$LAT (4) BYTE,
       OLD$LONG (4) BYTE,
       DELTA$X (4) BYTE,
       DELTA$Y (4) BYTE,
        I BYTE;
   DO I = 0 TO 3;
      OLD$LAT(I) = SYSTEM.LAT(I);
      OLD$LONG(I) = SYSTEM.LONG(I);
      END;
   CALL CRT$PRINT$STRING(.TITLE$9);
   CALL SEND$CRLF;
   CALL GET$LAT(.SYSTEM.LAT);
   CALL CRT$PRINT$STRING(.TITLE$9);
   CALL SEND$CRLF;
   CALL GET$LONG(.SYSTEM.LONG);
   CALL CONV$XY(.OLD$LAT, .OLD$LONG, .DELTA$X, .DELTA$Y);
   CALL TRANSLATE(.DELTA$X, .DELTA$Y, 1);
   CALL CLEAR$STRUCTURES;
   CALL SET$WINDOW;
   CALL PUT$OS$CENTER;
   CALL DRAW$EVERYTHING;
   CALL DISPLAY$PLASMA$SCALE;
   END ORIGIN;
```

```
/*****************************************************************************
*
* WIND:
*   THIS PROCEDURE IS USED TO GET INFORMATION ABOUT THE WIND.
*
*****************************************************************************/
WIND: PROCEDURE PUBLIC;
   DCL BUFFER(7) BYTE,
   (OK, TEMP) BYTE;
   DCL FP$360 (4) BYTE DATA (0DFH,0FFH,0B3H,043H);
   BUFFER(0) = 4;
   BUFFER(1) = 3;
   BUFFER(6) = 0;
   OK = 0;
   DO WHILE OK = 0;
      CALL CRT$PRINT$STRING(.TITLE$A);
      CALL SEND$CRLF;
      CALL CRT$PRINT$STRING(.('ENTER THE WIND DIRECTION AS REQUESTED:$$'));
      CALL SEND$CRLF;
      TEMP = 0;
      DO WHILE TEMP = 0;
         CALL CRT$PRINT$STRING(.('DEGREES: $$'));
         CALL PUT$NUMBER$BUFFER(3, .BUFFER(2));
         CALL CRT$WRITE('.');
         CALL PUT$NUMBER$BUFFER(1, .BUFFER(5));
         CALL ASCII$TO$FLOAT(.BUFFER, 7, .SYSTEM.WIND$DIR);
         TEMP = CHECK$FP$VALUE(.SYSTEM.WIND$DIR, .FP$360);
         END;
      CALL SEND$CRLF;
      OK = CHECK$INPUT;
      CALL CLEAR$LOW$SCREEN;
```

```
      END;
   BUFFER(0) = 3;
   BUFFER(1) = 2;
   BUFFER(5) = 0;
   OK = 0;
   DO WHILE OK =0;
      CALL CRT$PRINT$STRING(.TITLE$A);
      CALL SEND$CRLF;
      CALL CRT$PRINT$STRING(.('ENTER THE WIND SPEED AS REQUESTED:$$'));
      CALL SEND$CRLF;
      CALL CRT$PRINT$STRING(.('KNOTS: $$'));
      CALL PUT$NUMBER$BUFFER(2, .BUFFER(2));
      CALL CRT$WRITE('.');
      CALL PUT$NUMBER$BUFFER(1, .BUFFER(4));
      CALL ASCII$TO$FLOAT(.BUFFER, 6, .SYSTEM.WIND$SPD);
      CALL SEND$CRLF;
      OK = CHECK$INPUT;
      CALL CLEAR$LOW$SCREEN;
      END;
   END WIND;
```

```
/******************************************************************************
*
* SCALE:
*    THIS PROCEDURE IS USED TO UPDATE THE GRAPHICS SCALE VALUE.
*
******************************************************************************/
SCALE: PROCEDURE PUBLIC;
   CALL CRT$PRINT$STRING(.TITLE$B);
   CALL SEND$CRLF;
   CALL GET$SCALE(.SYSTEM.SCALE);
   CALL CLEAR$STRUCTURES;
   CALL SET$WINDOW;
   CALL PUT$OS$CENTER;
   CALL DRAW$EVERYTHING;
   CALL DISPLAY$PLASMA$SCALE;
   END SCALE;
```

```
/*****************************************************************
*
* GET$SAFE$RNG:
*    THIS PROCEDURE IS USED TO OBTAIN THE VALUE OF SAFE CPA RANGE USED TO
*    WARN THE OPERATOR THAT A CONTACT WILL BE IN COLLISION.
*
*****************************************************************/
GET$SAFE$RNG: PROCEDURE PUBLIC;
   DCL BUFFER (7) BYTE,
       (OK, TEMP, TEMP1) BYTE;
   DCL LOW$BOUND (4) BYTE DATA (008H, 03CH, 0CAH, 03CH), /* 0.024686827 */
       HIGH$BOUND (4) BYTE DATA (000H, 000H, 000H, 03FH),/* 0.5 */
       FP$2000 (4) BYTE DATA (0E4H, 02BH, 0FDH, 044H);    /* 2025.3716 */
   DCL M0 (*) BYTE DATA ('ENTER THE SAFE C.P.A. RANGE AS REQUESTED:$$'),
       M1 (*) BYTE DATA ('YARDS:  $$');

   OK = 0;
   DO WHILE OK = 0;
      CALL CRT$PRINT$STRING(.TITLE$D);
      CALL SEND$CRLF;
      CALL CRT$PRINT$STRING(.M0);
      CALL SEND$CRLF;
      TEMP, TEMP1 = 0;
      DO WHILE (TEMP = 0) OR (TEMP1 = 0);
         CALL CRT$PRINT$STRING(.M1);
         BUFFER(0), BUFFER(1) = 4;
         BUFFER(6) = 0;
         CALL PUT$NUMBER$BUFFER(4, .BUFFER(2));
         CALL ASCII$TO$FLOAT(.BUFFER, 7, .SAFE$RNG);
         CALL FDIV(.SAFE$RNG, .FP$2000, .SAFE$RNG);
         TEMP = CHECK$FP$VALUE(.SAFE$RNG, .HIGH$BOUND);
```

```
      IF TEMP <> 0
         THEN TEMP1 = CHECK$FP$VALUE(.LOW$BOUND, .SAFE$RNG);
      END;
   CALL SEND$CRLF;
   OK = CHECK$INPUT;
   CALL CLEAR$LOW$SCREEN;
   END;
END GET$SAFE$RNG;
```

Thesis
G5469
pt.2
c.1

177101

Goncalves

A microcomputer based shipboard surface-sub-surface contact plotter system.

4 OCT 78
30 JAN 79
9 NOV 79
14 MAY 81
[illegible] DEC 81
19 AUG 83
23 NOV 84

24924
24428
252[illegible]
26413
26413
2791[illegible]
27986

Thesis
G5469
pt.2
c.1

177101

Goncalves

A microcomputer based shipboard surface-sub-surface contact plotter system.

thesG5469pt.1
A microcomputer based shipboard surface-
3 2768 002 13083 3
DUDLEY KNOX LIBRARY